# 1 PRODUCT OVERVIEW

## SAM88RCRI PRODUCT FAMILY

Samsung's SAM88RCRI family of 8-bit single-chip CMOS microcontrollers offer fast and efficient CPU, a wide range of integrated peripherals, and supports OTP device.

A dual address/data bus architecture and bit- or nibble-configurable I/O ports provide a flexible programming environment for applications with varied memory and I/O requirements. Timer/counters with selectable operating modes are included to support real-time operations.

## S3C9654/C9658/P9658 MICROCONTROLLER

The S3C9654/C9658/P9658 microcontroller with USB function can be used in a wide range of general purpose applications. It is especially suitable for mouse or joystick controller and is available in 16, 18, 20-pin DIP and SOP package.

The S3C9654/C9658/P9658 single-chip 8-bit microcontroller is fabricated using an advanced CMOS process. It is built around the powerful SAM88RCRI CPU core.

Stop and Idle power-down modes were implemented to reduce power consumption. To increase on-chip register space, the size of the internal register file was logically expanded. The S3C9654/C9658/P9658 has 4/8 Kbytes of program memory on-chip (S3C9654/C9658), and 208 bytes of RAM including 16 bytes of working register.

Using the SAM88RCRI design approach, the following peripherals were integrated with the SAM88RCRI core:

— Three configurable I/O ports (14 pin, at 20 pin)

— 14-bit programmable pins for external interrupts (at 20 pin)

— 8-bit timer/counter with two operating modes

## OTP

The S3C9654/C9658 microcontroller is also available in OTP (One Time Programmable) version. S3P9658 microcontroller has an on-chip 4/8 Kbyte one-time-programmable EPROM instead of masked ROM. The S3P9658 is comparable to S3C9654/C9658, both in function and in pin configuration.

# FEATURES

### CPU

- SAM88RCRI CPU core

### Memory

- 4-K byte internal program memory (ROM S3C9654)
- 8-K byte internal program memory (ROM S3P9658/C9658)
- 208-byte RAM
- 16 bytes of working register

### Instruction Set

- 41 instructions
- IDLE and STOP instructions added for power-down modes

### Instruction Execution Time

- 0.66 μs at 6 MHz $f_{OSC}$

### Interrupts

- 14 interrupt sources with one vector (20 pin)
- 12 interrupt sources with one vector (18 pin)
- 10 interrupt sources with one vector (16 pin)
- One level, one vector interrupt structure

### Oscillation Circuit Options

- 6 MHz crystal/ceramic oscillator
- External clock source
- RC oscillator
- Embedded oscillation capacitor (XI, XO, 33pF)

### General I/O

- 14 bit-programmable I/O pins (20 pin)
- 12 bit-programmable I/O pins (18 pin)
- 10 bit-programmable I/O pins (16 pin)

### Sub Oscillator

- Internal RC sub oscillator
- Auto interrupt wake-up

### Timer/Counter

- One 8-bit basic timer for watchdog function and programmable oscillation stabilization interval generation function
- One 8-bit timer/counter with Compare/Overflow counter

### USB Serial Bus

- Compatible to USB low speed (1.5 Mbps) device 1.0 specification.
- Serial bus interface engine (SIE)
  - — Packet decoding/generation
  - — CRC generation and checking
  - — NRZI encoding/decoding and bit-stuffing
- Two 8-byte receive/transmit USB buffer

### Operating Temperature Range

- – 0°C to + 85°C

### Operating Voltage Range

- 4.0 V to 5.25 V

### Package Types

- 16, 18, 20 pin DIP
- 16, 18, 20 pin SOP

### Comparator

- 6-channel mode, 32 step resolution
- 5-channel mode, external reference
- Low EMI design

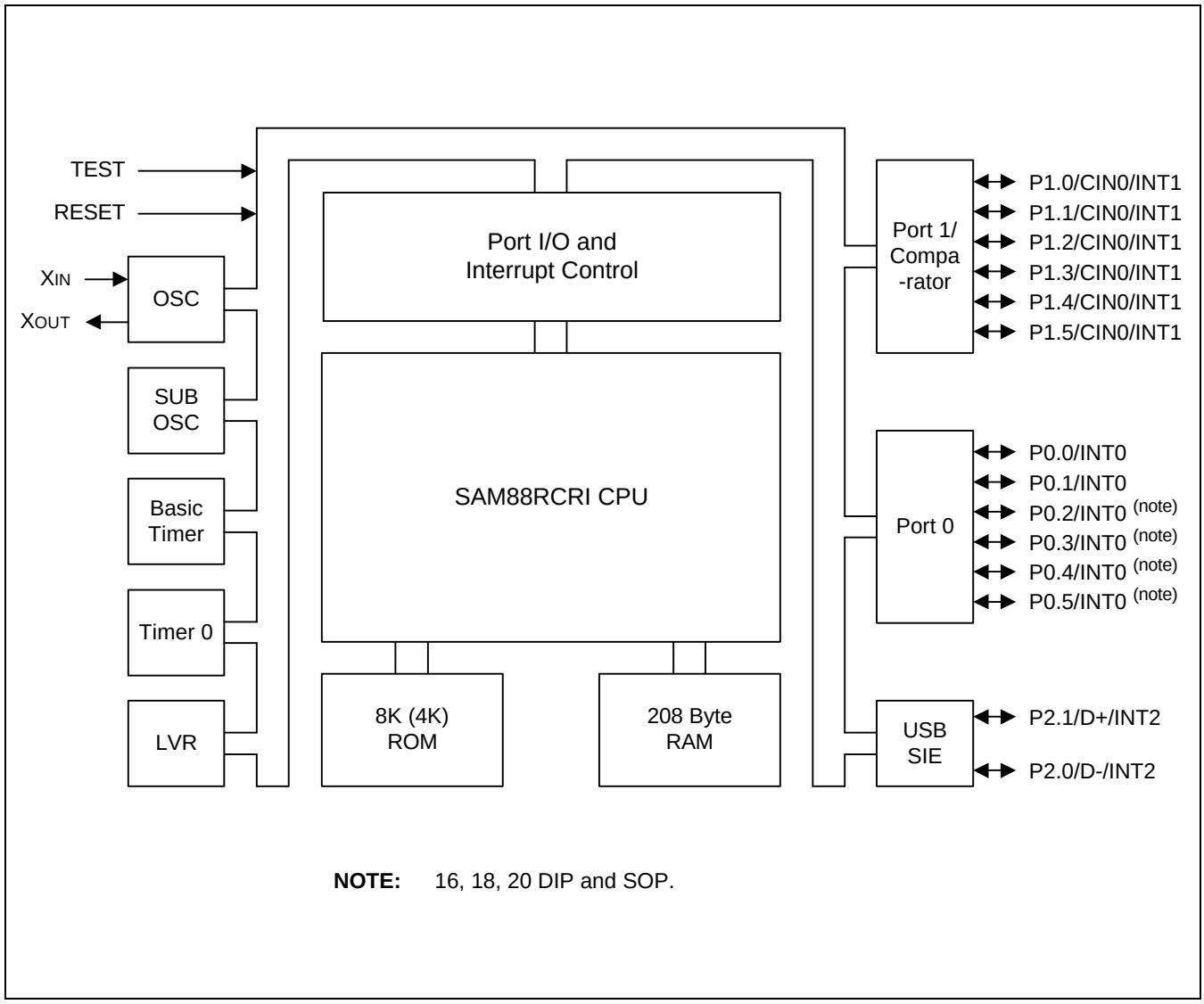
### Low Voltage Reset

- Low voltage Reset
- Power on Reset

### High Sink Current Pin for LED

- P0.0 ($V_{OL}$: 0.4 V, 50mA)

## BLOCK DIAGRAM




**NOTE:** 16, 18, 20 DIP and SOP.

**Figure 1-1. Block Diagram**

## PIN ASSIGNMENTS

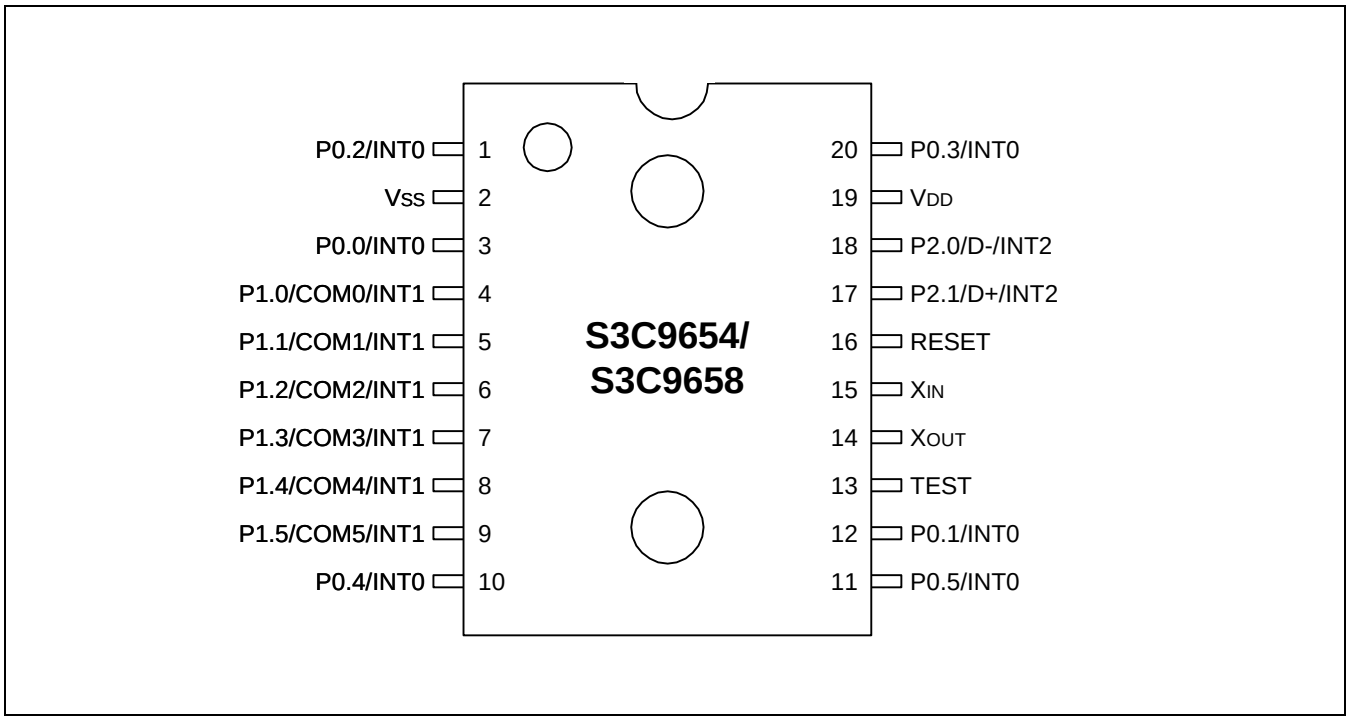



**Figure 1-2. Pin Assignment (20 Pin)**

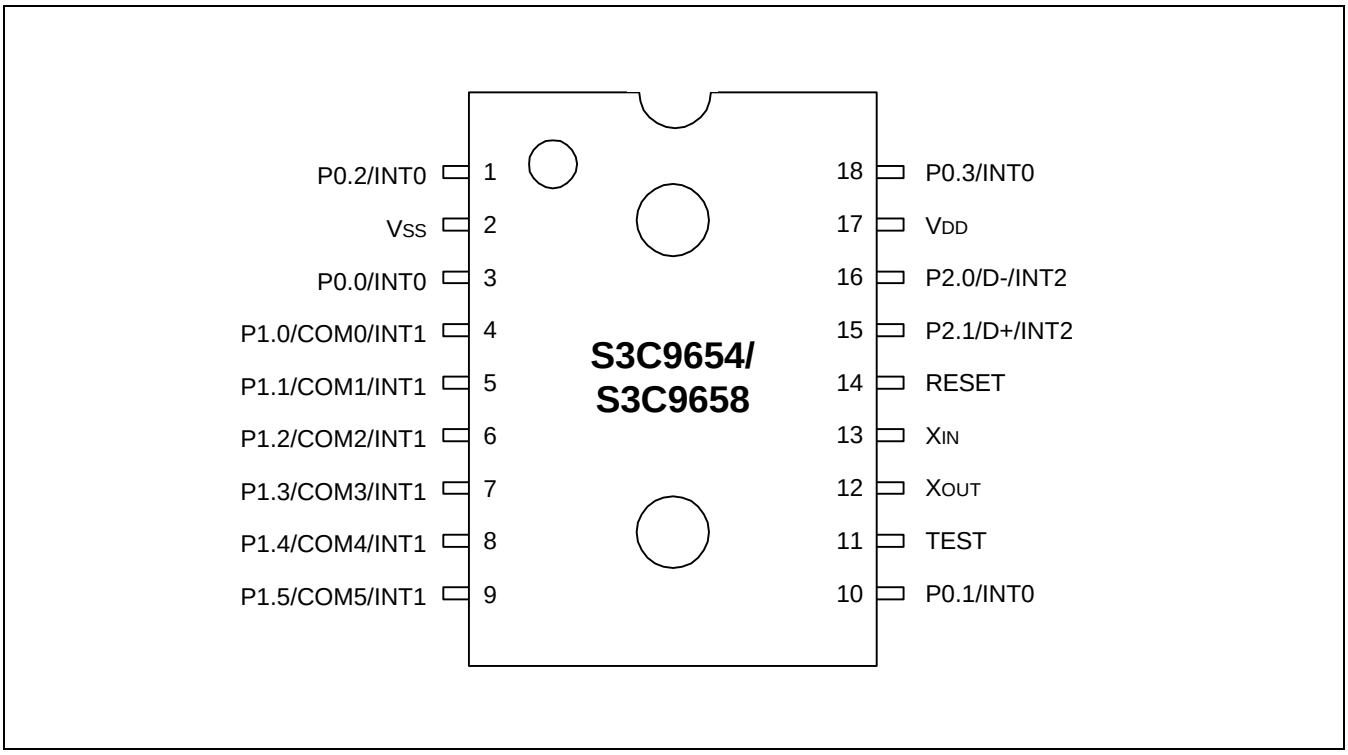



**Figure 1-3. Pin Assignment (18 Pin)**

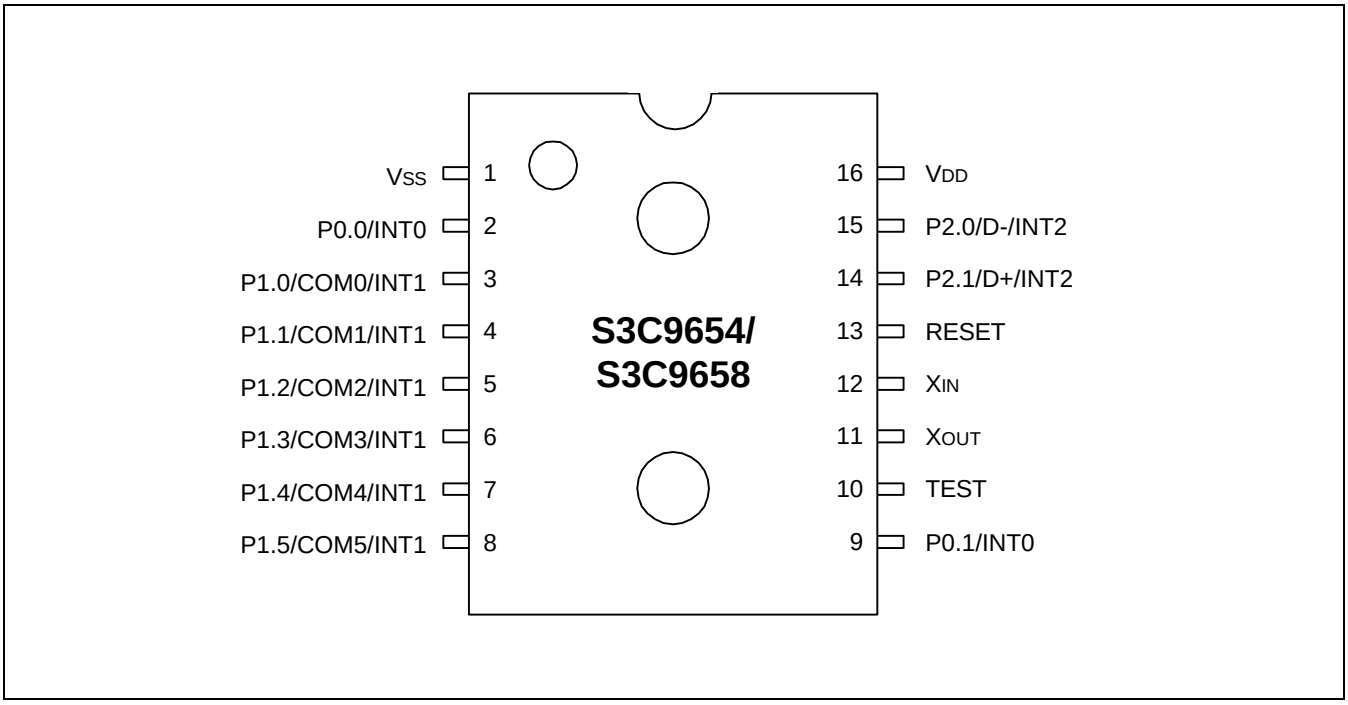



**Figure 1-4. Pin Assignment (16 Pin)**

**Table 1-1. Signal Descriptions**

| Pin Names | Pin Type | Pin Description | Circuit Number | Pin Numbers | Share Pins |
|---|---|---|---|---|---|
| P0.0 | I/O | Bit-programmable I/O port for Schmitt trigger input or n-ch open drain output (50 mA). Pull-up resistor is assignable to input pin by software and is automatically disabled for output pin. Port 0 can be individually configured as external interrupt input. | SK | 3 | INT0 |
| P0.1–P0.5 | I/O | Bit-programmable I/O port for Schmitt trigger input or push-pull output. Pull-up resistors individually assignable to input pins by software and are automatically disabled for output pins. Port 0 can be individually configured as external interrupt inputs. | D | 1, 10, 11, 12, 20 | INT0 |
| P1.0–P1.5 | I/O | Bit-programmable I/O port for Schmitt trigger input or push-pull output. Pull-up resistors are individually assignable to input pins by software. Port 1 can be configured as comparator input or external interrupt inputs. Pull-down resistors are individually assignable. (in comparator input) | CP | 4–9 | CIN0-5 INT1 |
| P2.0/D- – P2.1/D+ | I/O | Bit-programmable I/O port for Schmitt trigger input or n-ch open drain output. Pull-up resistors are individually assignable to input pins by software and are automatically disabled for output pins. Port 2 can be individually configured as external interrupt inputs. Also it can be configured as an USB ports. | CP | 17, 18 | INT2 |
| $X_{OUT}$, $X_{IN}$ | – | System clock input and output pin (crystal/ceramic oscillator, or external clock source) | – | 14, 15 | – |
| INT0 | I | External interrupt for bit-programmable port 0 | D | 1, 3, 10, 11, 12, 20 | Port 0 |
| INT1 | I | External interrupt for bit-programmable port 1 | D | 4–9 | Port 1 |
| INT2 | I | External interrupt for bit-programmable port 2 | D | 17, 18 | Port 2 |
| $V_{DD}$ | – | Power input pin | – | 19 | – |
| $V_{SS}$ | – | $V_{SS}$ is a ground power for CPU core. | – | 2 | – |
| RESET | 1 | Reset input pin (Pull-up register embedded) | – | 16 | – |

**Table 1-2. Pin Circuit Assignments for the S3C9654/C9658/P9658**

| Circuit Number | Circuit Type | S3C9654/C9658/P9658 Assignments |
|---|---|---|
| C | O | |
| D | I/O | Port 0.1–5, INT0, INT1, INT2 |
| SK | I/O | Port 0.0 |
| CP | I/O | Port 1, Port 2 |

**NOTE:** Diagrams of circuit types C–D, and F-8 are presented below.

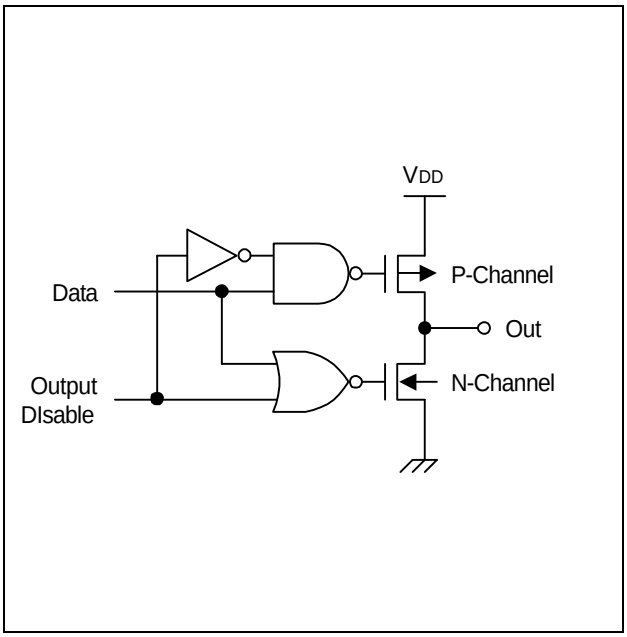



**Figure 1-5. Pin Circuit Type C**

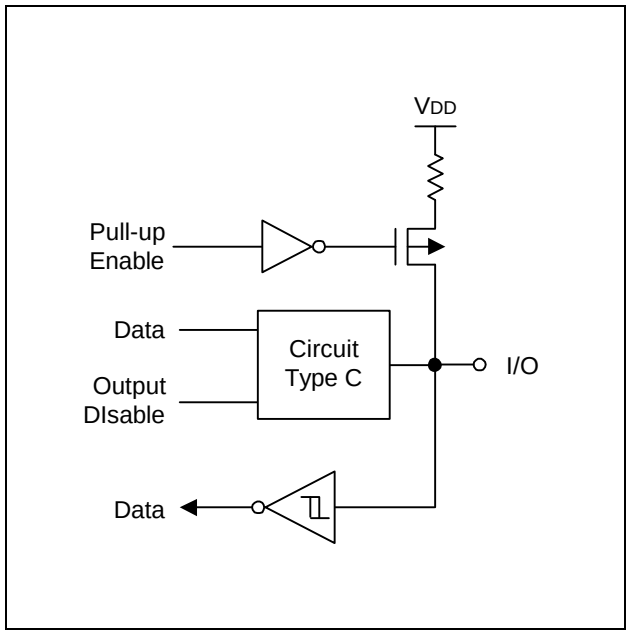
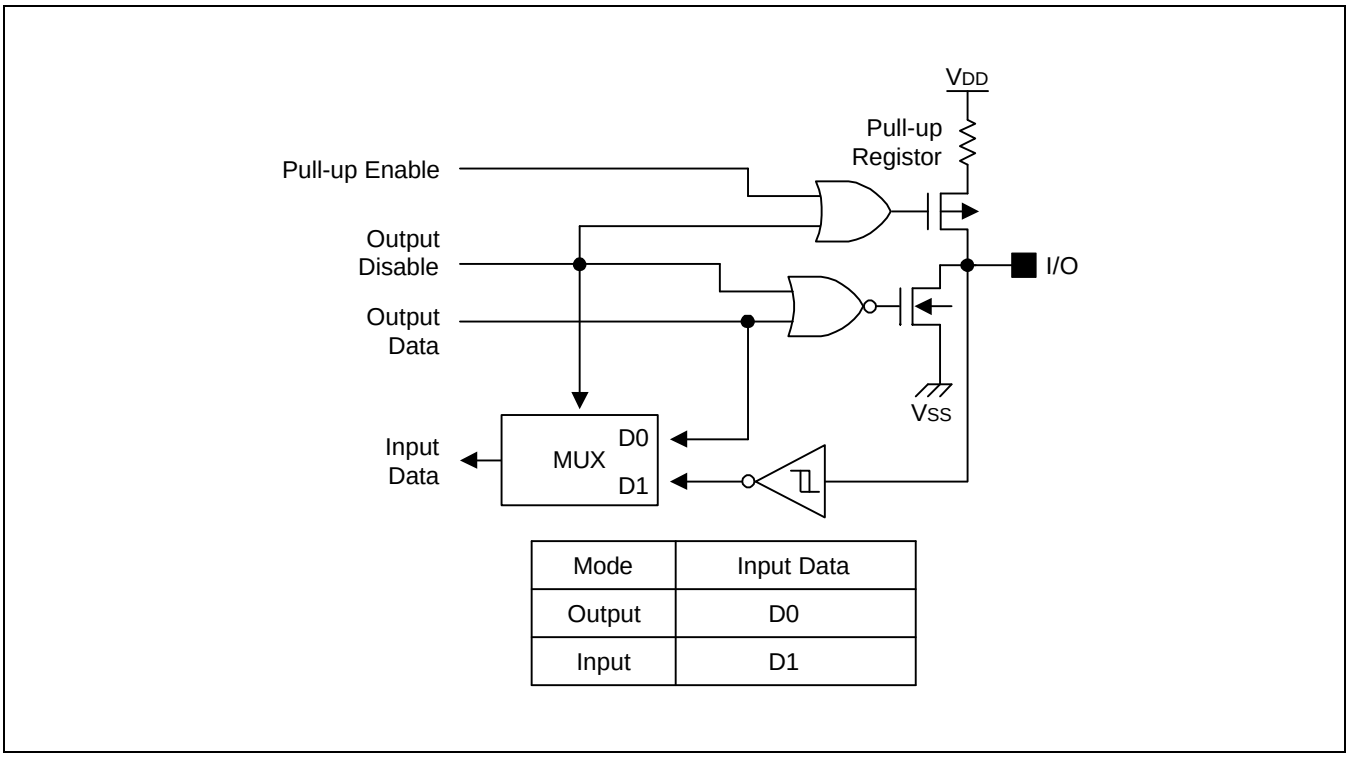



**Figure 1-6. Pin Circuit Type D**




| Mode | Input Data |
|---|---|
| Output | D0 |
| Input | D1 |

**Figure 1-7. Pin Circuit Type SK**

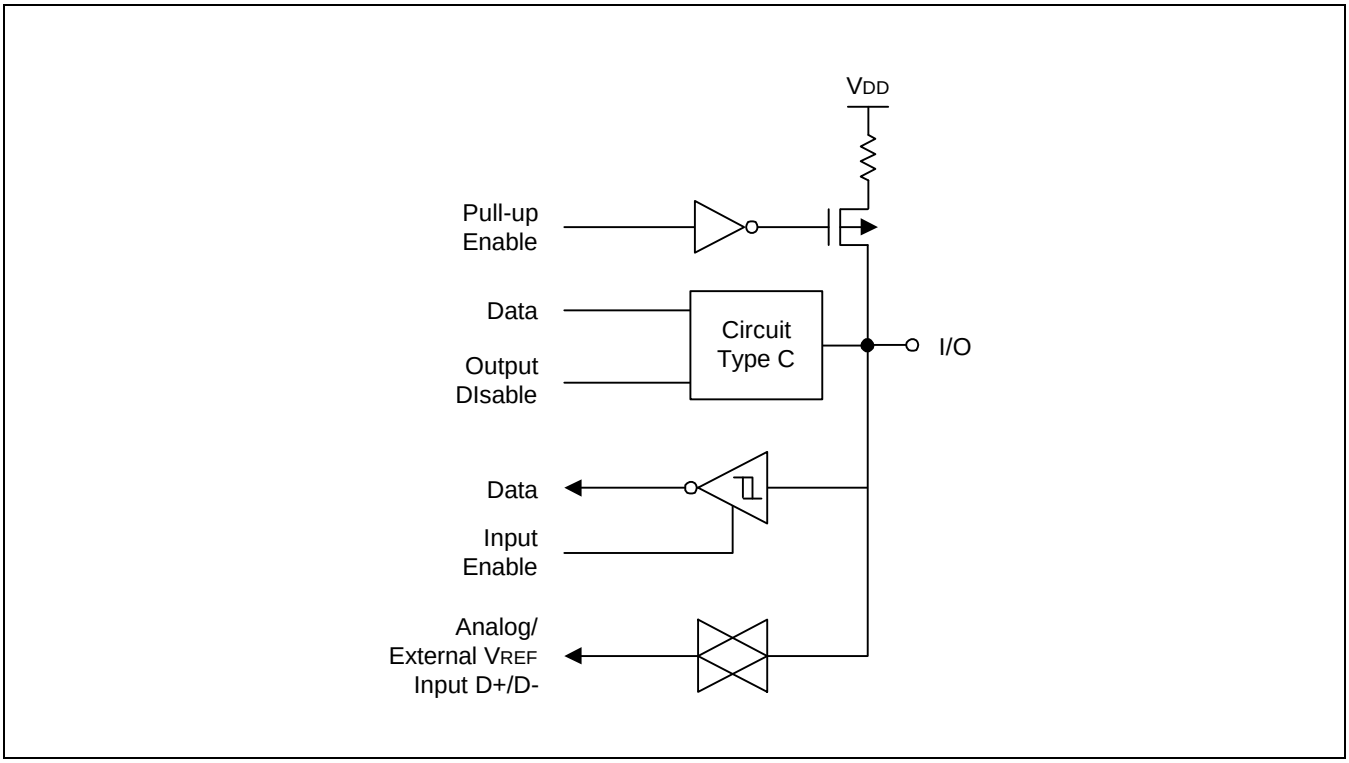
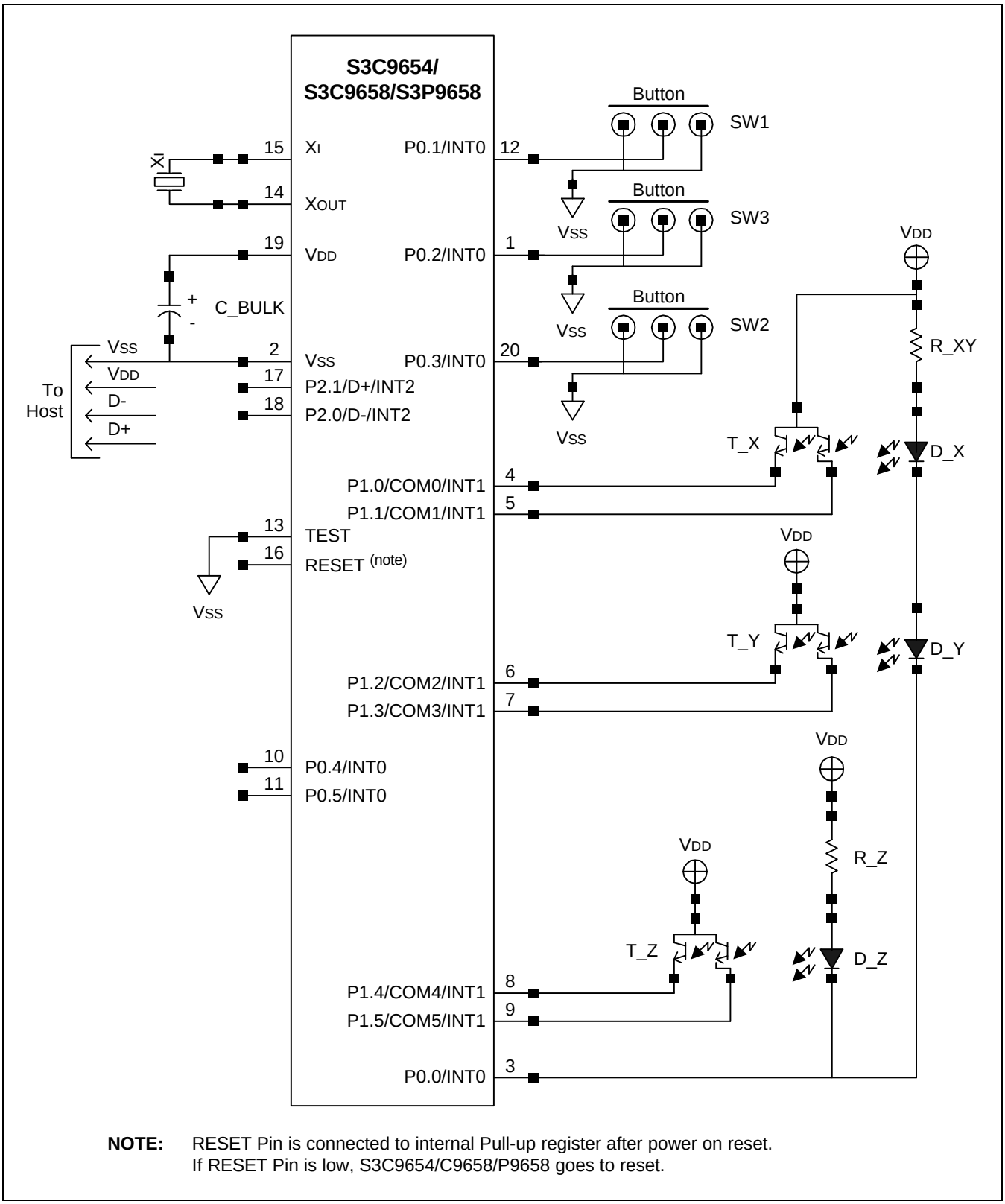



**Figure 1-8. Pin Circuit Type CP**




**NOTE:** RESET Pin is connected to internal Pull-up register after power on reset.
If RESET Pin is low, S3C9654/C9658/P9658 goes to reset.

**Figure 1-9. USB Mouse Circuit Diagram**

# NOTES

# 2 ADDRESS SPACES

## OVERVIEW

The S3C9654/C9658/P9658 microcontroller has two kinds of address space:

— Program memory (ROM)
— Internal register file

A 13-bit address bus supports both program memory. Special instructions and related internal logic determine when the 13-bit bus carries addresses for program memory. A separate 8-bit register bus carries addresses and data between the CPU and the internal register file.

The S3C9654/C9658 has 4/8 Kbytes of mask-programmable program memory on-chip. The S3C9654/C9658/P9658 microcontroller has 192 bytes general-purpose registers in its internal register file. Forty-eight bytes in the register file are mapped for system and peripheral control functions.

# PROGRAM MEMORY (ROM)

## NORMAL OPERATING MODE (INTERNAL ROM)

The S3C9654/C9658/P9658 has 4/8 Kbytes of internal mask-programmable program memory. The first 2 bytes of the ROM (0000H–0001H) are an interrupt vector address. The program reset address in the ROM is 0100H.

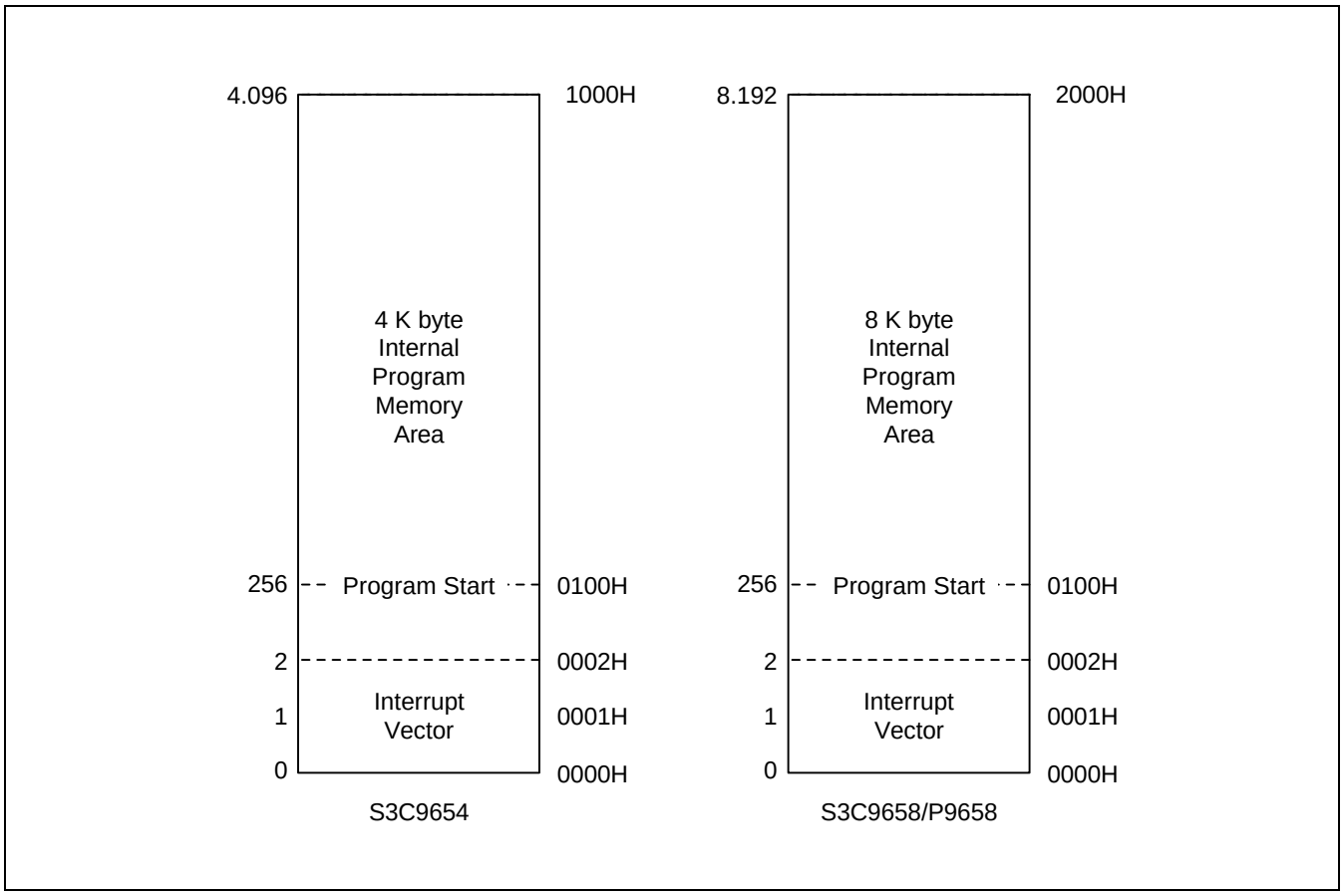



**Figure 2-1. S3C9654/C9658/P9658 Program Memory Address Space**

## REGISTER ARCHITECTURE

The upper 64 bytes of the S3C9654/C9658/P9658's internal register file are addressed as working registers, system control registers and peripheral control registers. The lower 192 bytes of internal register file (00H–BFH) is called the general purpose register space.

For many SAM88RCRI microcontrollers, the addressable area of the internal register file is further expanded by the additional of one or more register pages at general purpose register space (00H–BFH). This register file expansion is not implemented in the S3C9654/C9658/P9658.

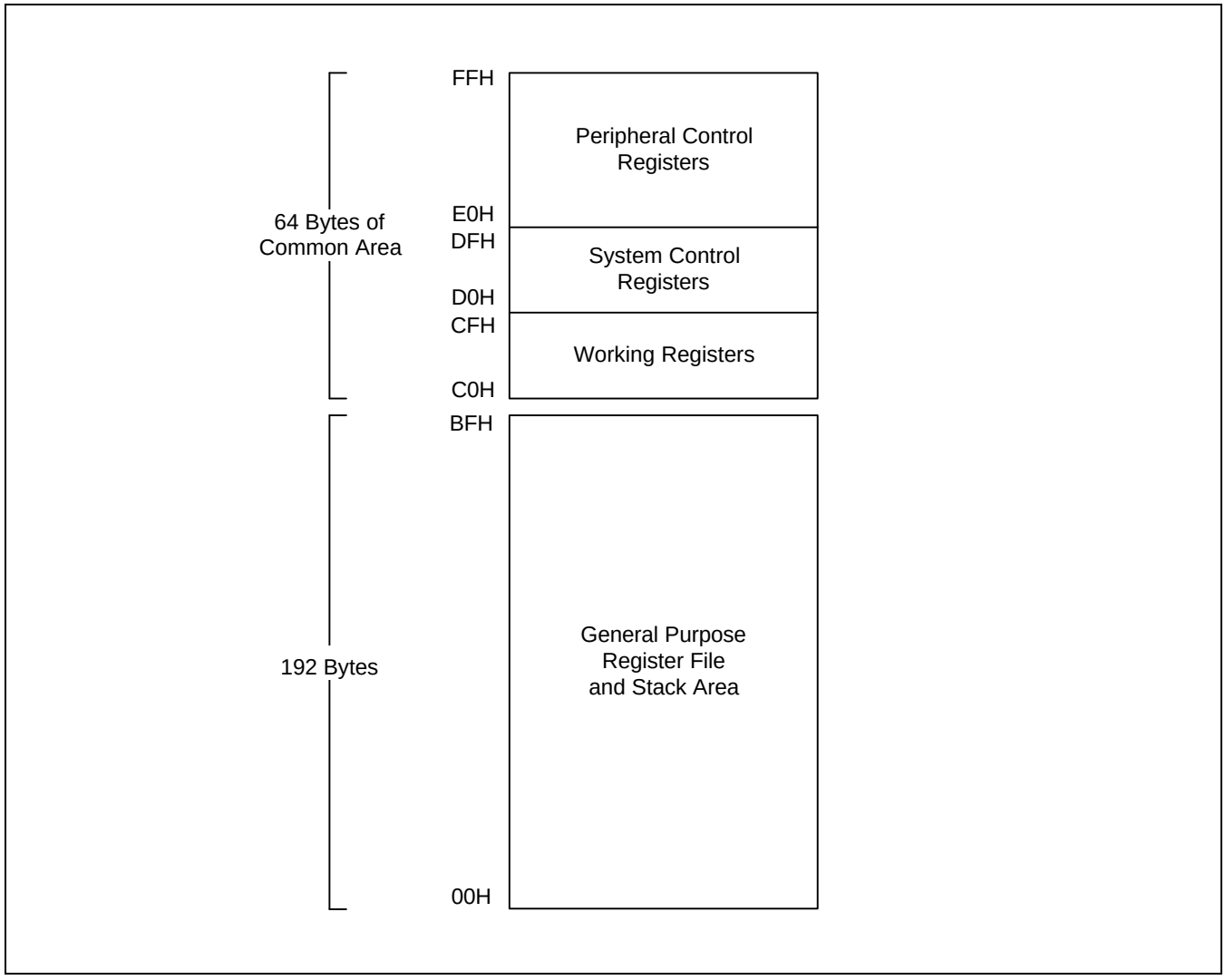



**Figure 2-2. Internal Register File Organization**

## COMMON WORKING REGISTER AREA (C0H–CFH)

The SAM88RCRI register architecture provides an efficient method of working register addressing that takes full advantage of shorter instruction formats to reduce execution time.

This16-byte address range is called common area. That is, locations in this area can be used as working registers by operations that address any location on any page in the register file. Typically, these working registers serve as temporary buffers for data operations between different pages. However, because the S3C9654/C9658/P9658 uses only page 0, you can use the common area for any internal data operation.

The Register (R) addressing mode can be used to access this area

Registers are addressed either as a single 8-bit register or as a paired 16-bit register. In 16-bit register pairs, the address of the first 8-bit register is always an even number and the address of the next register is an odd number. The most significant byte of the 16-bit data is always stored in the even-numbered register; the least significant byte is always stored in the next (+ 1) odd-numbered register.

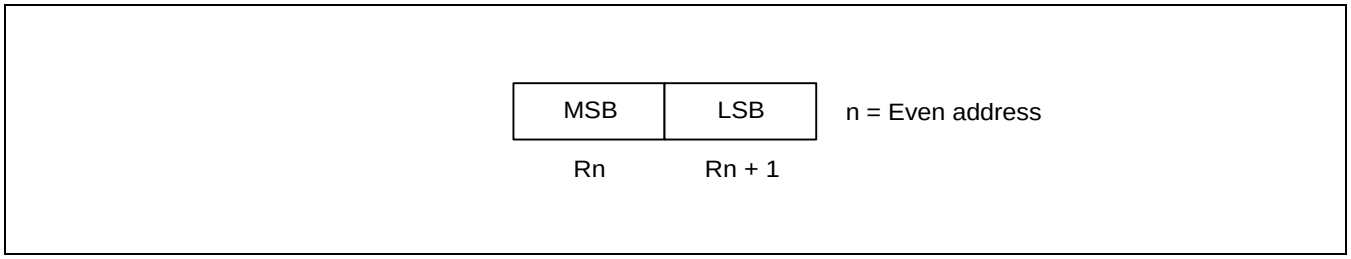



**Figure 2-3. 16-Bit Register Pairs**

### ☞ PROGRAMMING TIP — Addressing the Common Working Register Area

As the following examples show, you should access working registers in the common area, locations C0H–CFH, using working register addressing mode only.

Examples:

1. LD 0C2H,40H ; Invalid addressing mode!

   Use working register addressing instead:

   LD R2,40H ; R2 (C2H) ← the value in location 40H

2. ADD 0C3H,#45H ; Invalid addressing mode!

   Use working register addressing instead:

   ADD R3,#45H ; R3 (C3H) ← R3 + 45H

## SYSTEM STACK

KS86-series microcontrollers use the system stack for subroutine calls and returns and to store data. The PUSH and POP instructions are used to control system stack operations. The S3C9654/C9658/P9658 architecture supports stack operations in the internal register file.

### STACK OPERATIONS

Return addresses for procedure calls and interrupts and data are stored on the stack. The contents of the PC are saved to stack by a CALL instruction and restored by the RET instruction. When an interrupt occurs, the contents of the PC and the FLAGS register are pushed to the stack. The IRET instruction then pops these values back to their original locations. The stack address is always decremented before a push operation and incremented after a pop operation. The stack pointer (SP) always points to the stack frame stored on the top of the stack, as shown in Figure 2-4.

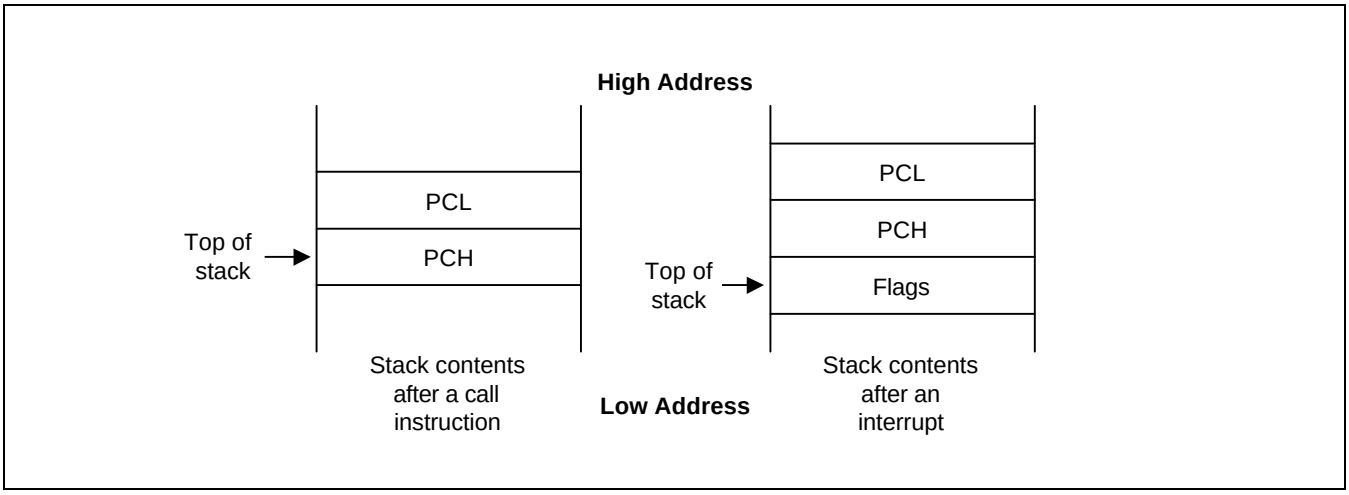



**Figure 2-4. Stack Operations**

### STACK POINTER (SP)

Register location D9H contains the 8-bit stack pointer (SP) that is used for system stack operations. After a reset, the SP value is undetermined.

Because only internal memory space is implemented in the KS86C6104/P6104, the SP must be initialized to an 8-bit value in the range 00H–BFH.

**NOTE**

In case a Stack Pointer is initialized to 00H, it is decreased to FFH when stack operation starts. This means that a Stack Pointer access invalid stack area.

### ☞ PROGRAMMING TIP — Standard Stack Operations Using PUSH and POP

The following example shows you how to perform stack operations in the internal register file using PUSH and POP instructions:

```
LD      SP,#0C0H        ; SP ← C0H (Normally, the SP is set to 0C0H by the
                        ; initialization routine)
•
•
•
PUSH    SYM             ; Stack address 0BFH ← SYM
PUSH    CCON            ; Stack address 0BEH ← CCON
PUSH    20H             ; Stack address 0BDH ← 20H
PUSH    R3              ; Stack address 0BCH ← R3
•
•
•
POP     R3              ; R3 ← Stack address 0BCH
POP     20H             ; 20H ← Stack address 0BDH
POP     CCON            ; CCON ← Stack address 0BEH
POP     SYM             ; SYM ← Stack address 0BFH
```

# 3 ADDRESSING MODES

## OVERVIEW

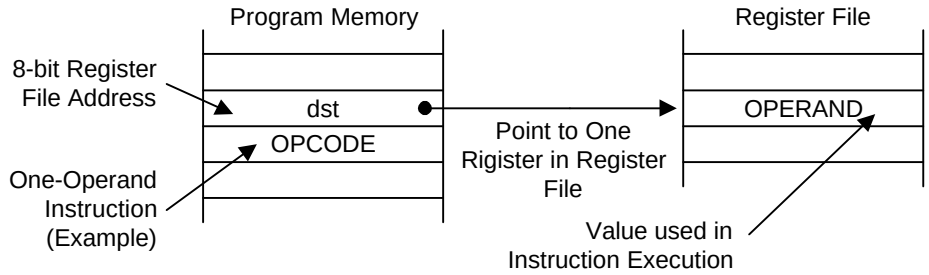
Instructions that are stored in program memory are fetched for execution using the program counter. Instructions indicate the operation to be performed and the data to be operated on. Addressing mode is the method used to determine the location of the data operand. The operands specified in SAM88RCRI instructions may be condition codes, immediate data, or a location in the register file, program memory, or data memory.

The SAM88RCRI instruction set supports six explicit addressing modes. Not all of these addressing modes are available for each instruction. The addressing modes and their symbols are as follows:

— Register (R)
— Indirect Register (IR)
— Indexed (X)
— Direct Address (DA)
— Relative Address (RA)
— Immediate (IM)

## REGISTER ADDRESSING MODE (R)

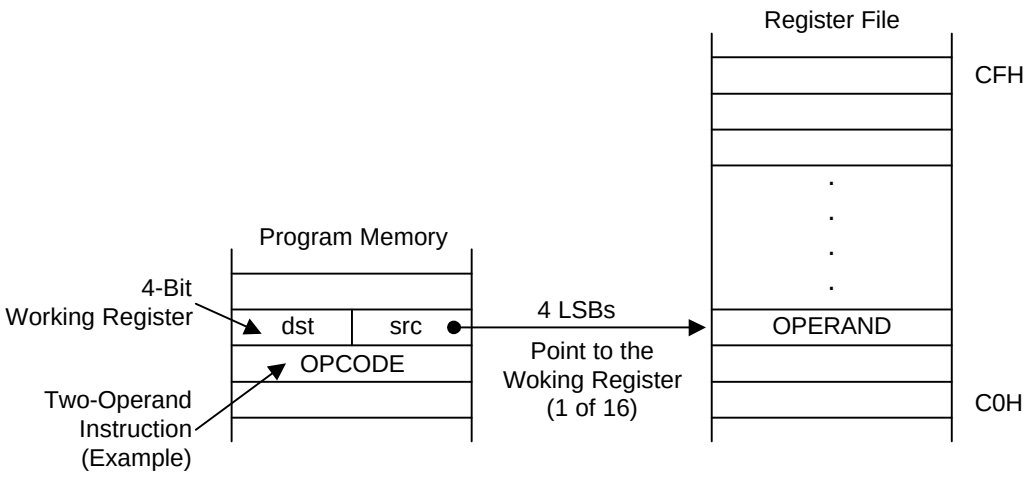
In Register addressing mode, the operand is the content of a specified register (see Figure 3-1). Working register addressing differs from Register addressing because it uses a 16-byte working register space in the register file and a 4-bit register within that space (see Figure 3-2).




Sample Instruction:

```
DEC     CNTR            ;   Where CNTR is the label of an 8-bit register address
```

**Figure 3-1. Register Addressing**




Sample Instruction:

```
ADD     R1, R2          ;   Where R1 = C1H and R2 = C2H
```

**Figure 3-2. Working Register Addressing**

## INDIRECT REGISTER ADDRESSING MODE (IR)

In Indirect Register (IR) addressing mode, the content of the specified register or register pair is the address of the operand. Depending on the instruction used, the actual address may point to a register in the register file, to program memory (ROM), or to an external memory space (see Figures 3-3 through 3-6).

You can use any 8-bit register to indirectly address another register. Any 16-bit register pair can be used to indirectly address another memory location.

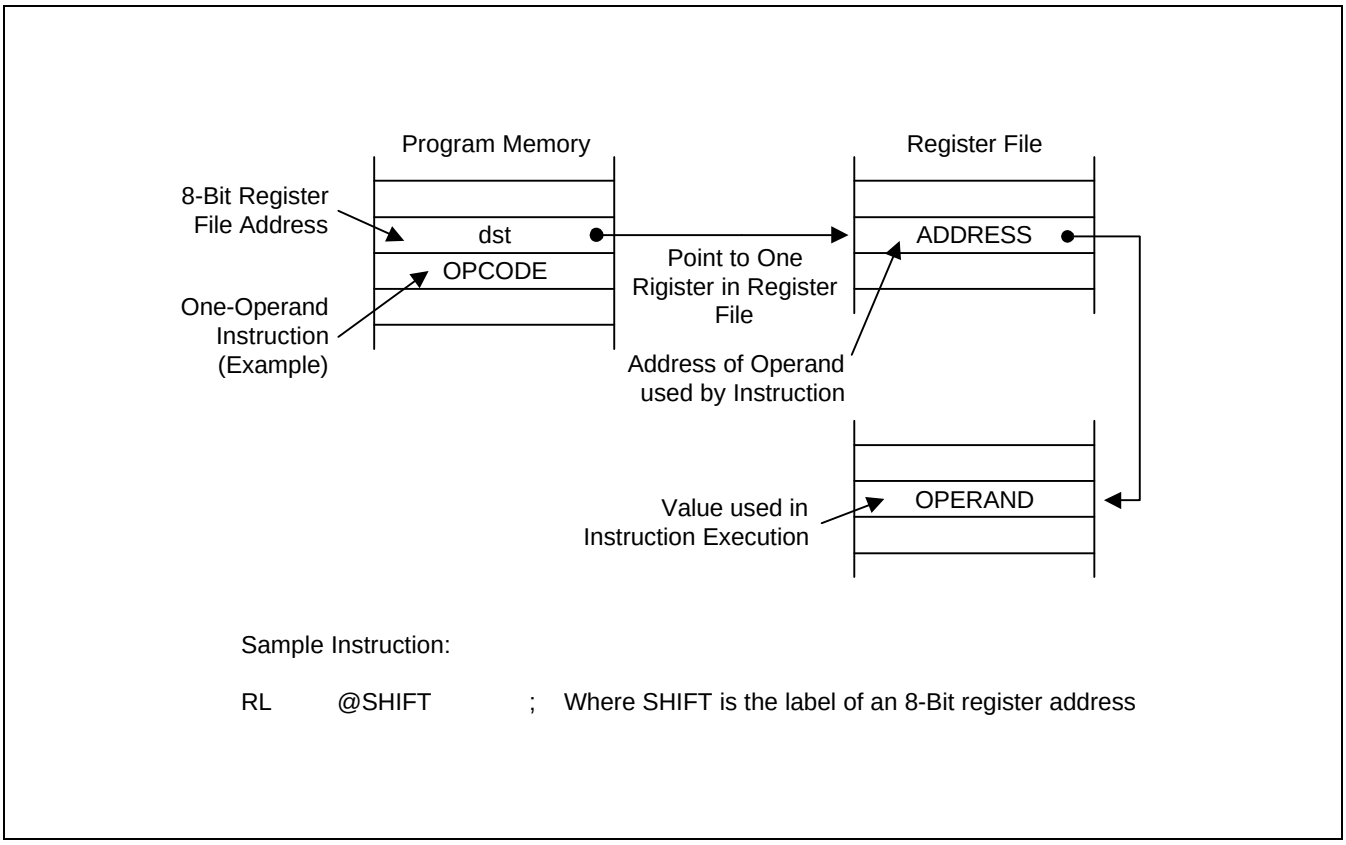



**Figure 3-3. Indirect Register Addressing to Register File**

## INDIRECT REGISTER ADDRESSING MODE (Continued)

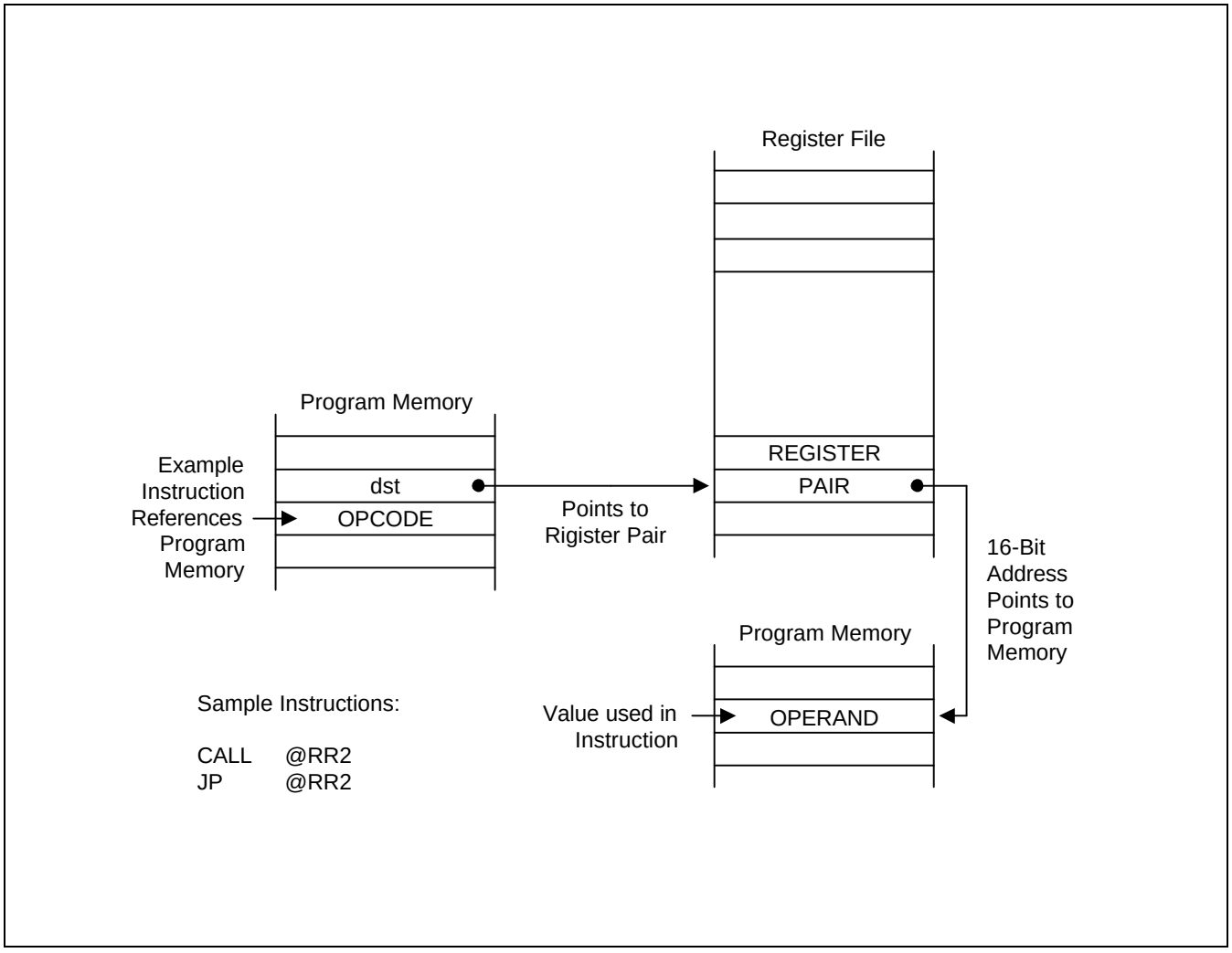



**Figure 3-4. Indirect Register Addressing to Program Memory**

## INDIRECT REGISTER ADDRESSING MODE (Continued)

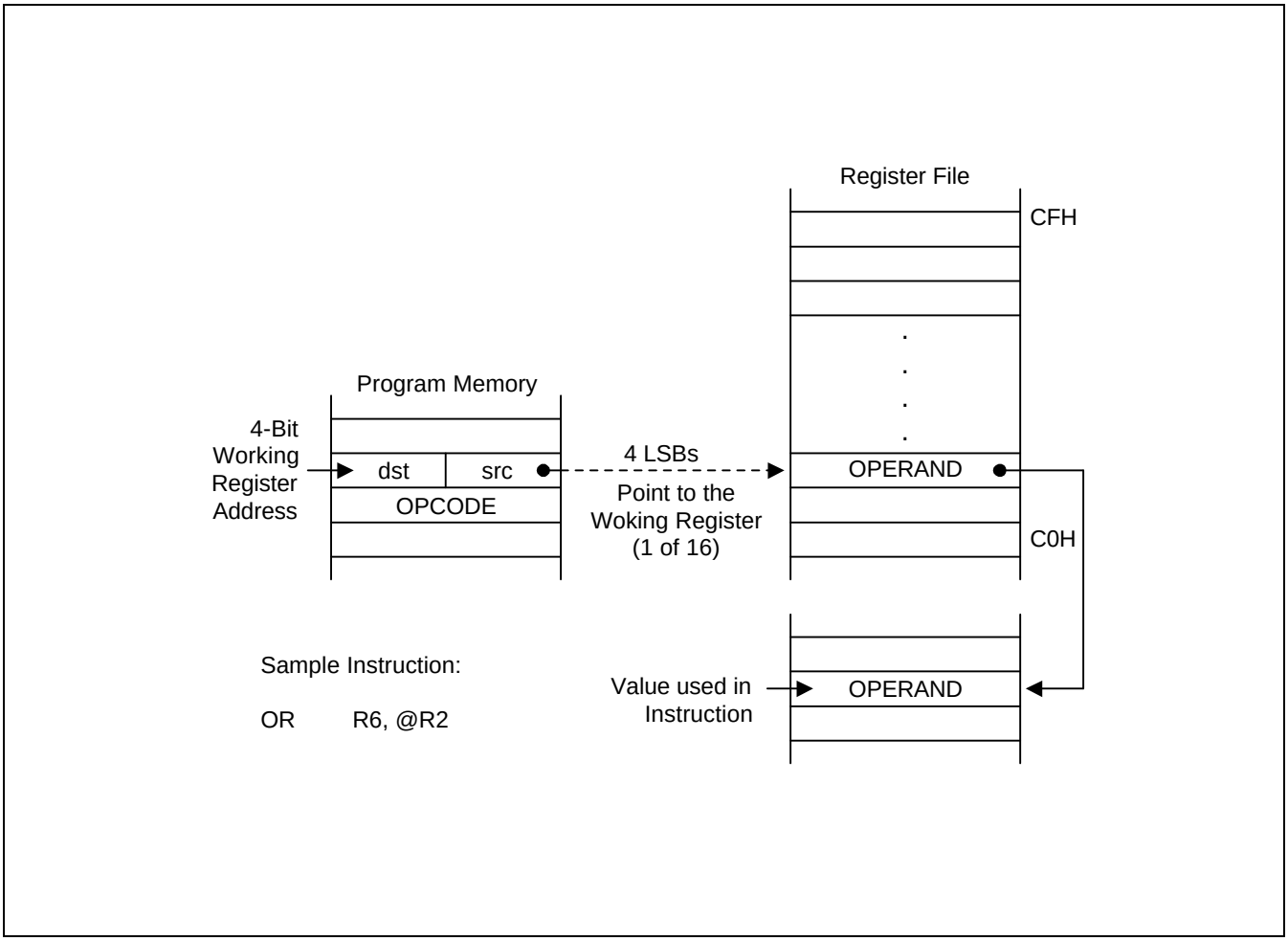
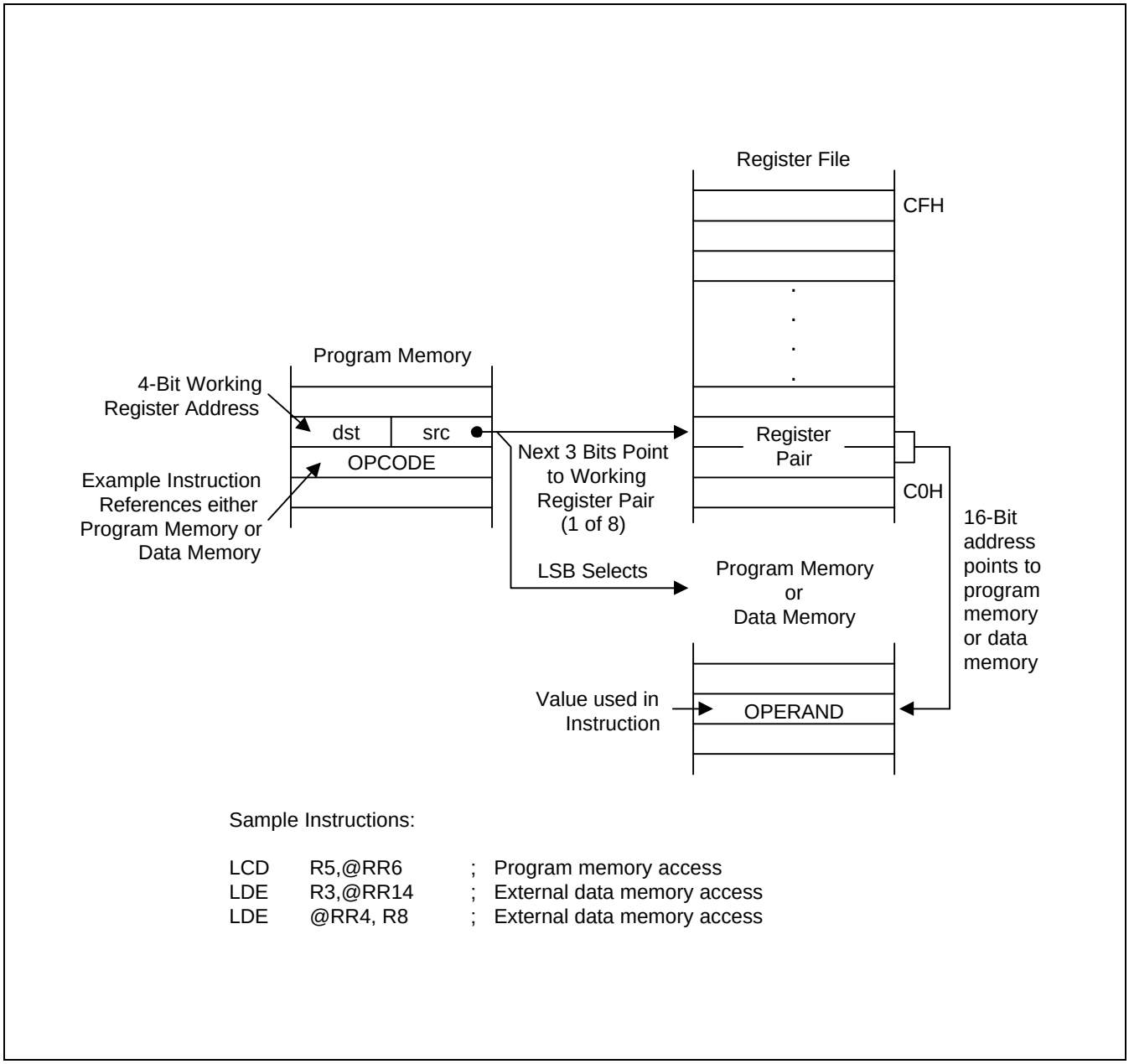



**Figure 3-5. Indirect Working Register Addressing to Register File**

## INDIRECT REGISTER ADDRESSING MODE (Concluded)




Sample Instructions:

```
LCD     R5,@RR6      ;  Program memory access
LDE     R3,@RR14     ;  External data memory access
LDE     @RR4, R8     ;  External data memory access
```

**Figure 3-6. Indirect Working Register Addressing to Program or Data Memory**

## INDEXED ADDRESSING MODE (X)

Indexed (X) addressing mode adds an offset value to a base address during instruction execution in order to calculate the effective operand address (see Figure 3-7). You can use Indexed addressing mode to access locations in the internal register file or in external memory.

In short offset Indexed addressing mode, the 8-bit displacement is treated as a signed integer in the range of –128 to +127. This applies to external memory accesses only (see Figure 3-8).

For register file addressing, an 8-bit base address provided by the instruction is added to an 8-bit offset contained in a working register. For external memory accesses, the base address is stored in the working register pair designated in the instruction. The 8-bit or 16-bit offset given in the instruction is then added to the base address (see Figure 3-9).

The only instruction that supports Indexed addressing mode for the internal register file is the Load instruction (LD). The LDC and LDE instructions support Indexed addressing mode for internal program memory, external program memory, and for external data memory, when implemented.

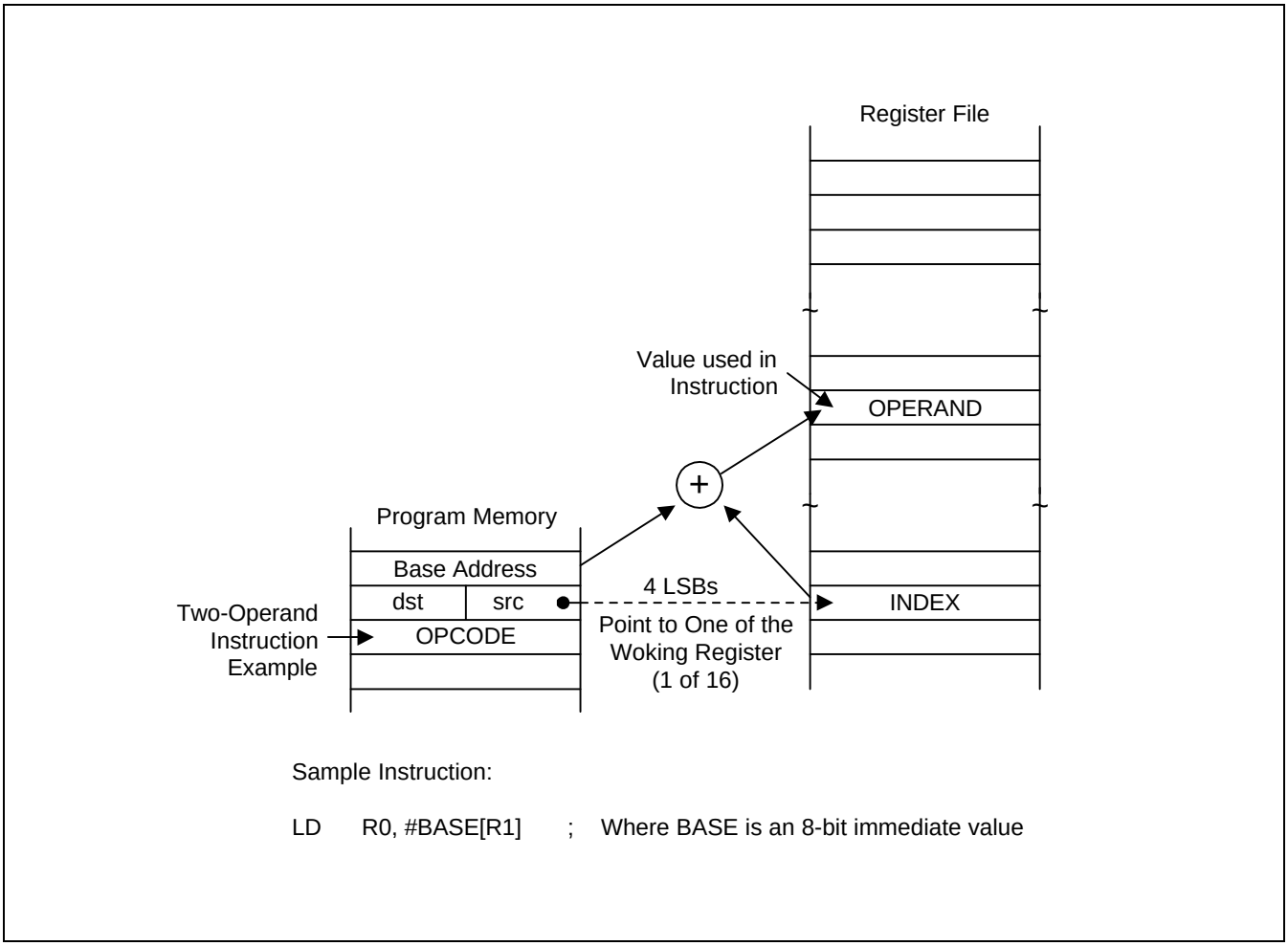
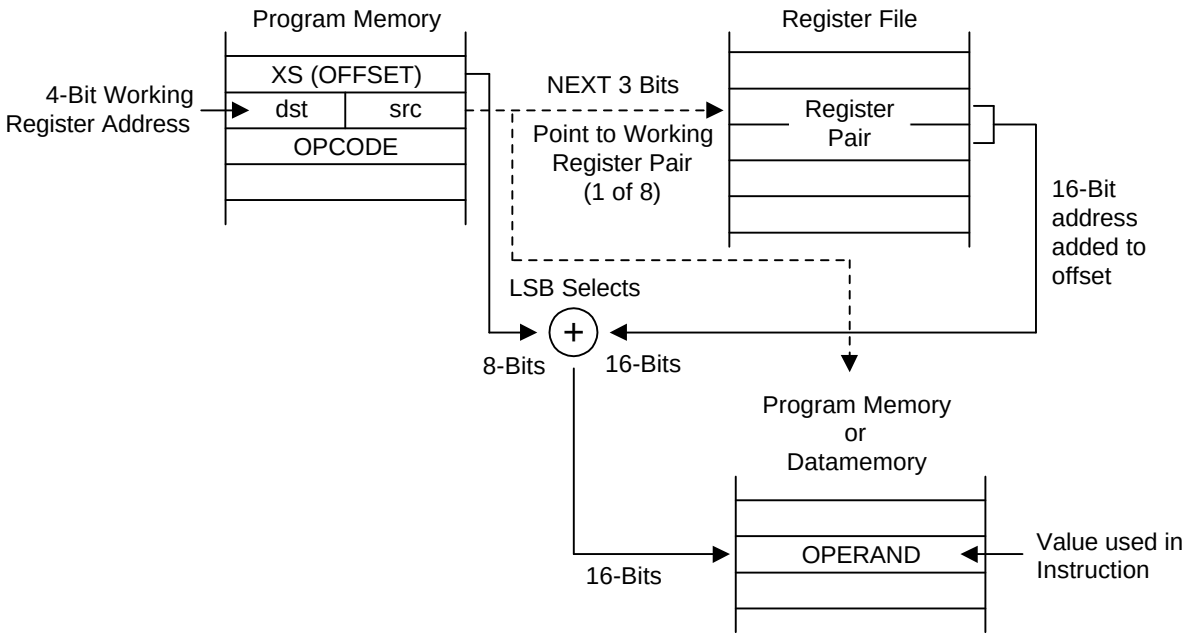



**Figure 3-7. Indexed Addressing to Register File**

## INDEXED ADDRESSING MODE (Continued)




Sample Instructions:

```
LDC     R4, #04H[RR2]       ;  The values in the program address (RR2 + #04H)
                               are loaded into register R4.
LDE     R4,#04H[RR2]        ;  Identical operation to LDC example, except that
                               external program memory is accessed.
```

**Figure 3-8. Indexed Addressing to Program or Data Memory with Short Offset**

## INDEXED ADDRESSING MODE (Concluded)

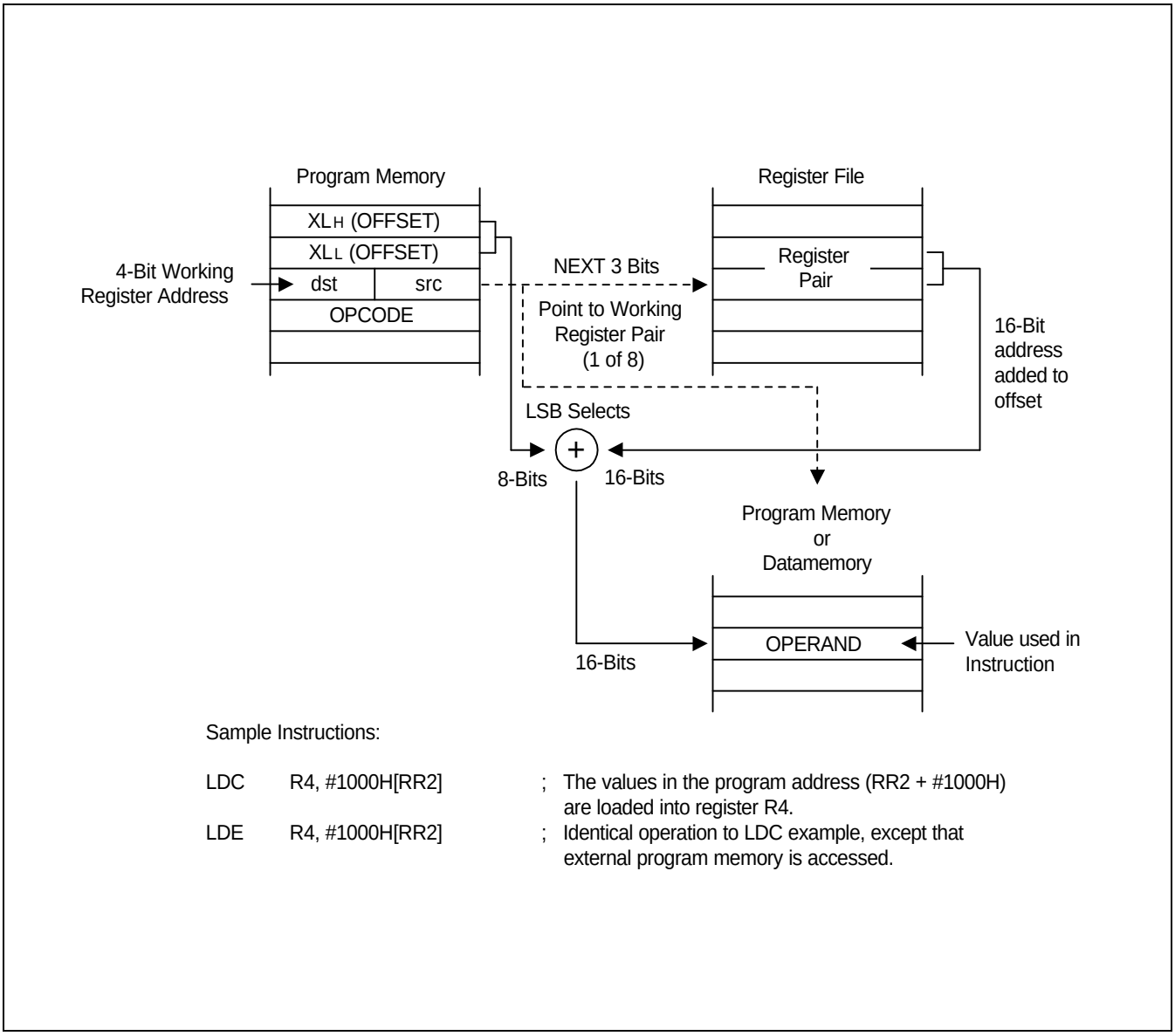
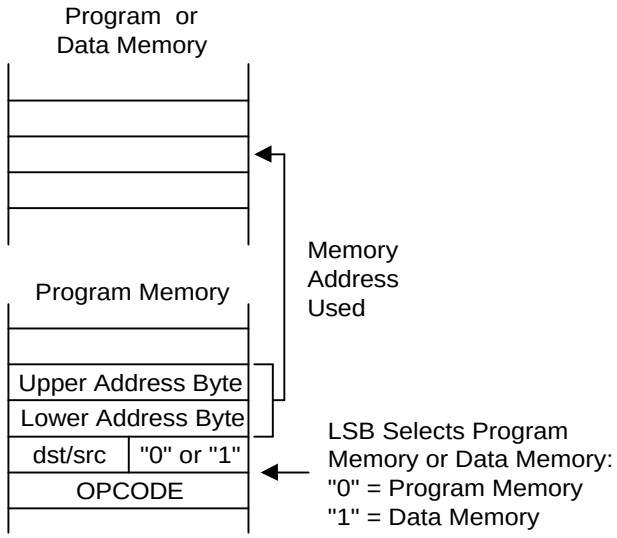



**Figure 3-9. Indexed Addressing to Program or Data Memory with Long Offset**

## DIRECT ADDRESS MODE (DA)

In Direct Address (DA) mode, the instruction provides the operand's 16-bit memory address. Jump (JP) and Call (CALL) instructions use this addressing mode to specify the 16-bit destination address that is loaded into the PC whenever a JP or CALL instruction is executed.

The LDC and LDE instructions can use Direct Address mode to specify the source or destination address for Load operations to program memory (LDC) or to external data memory (LDE), if implemented.

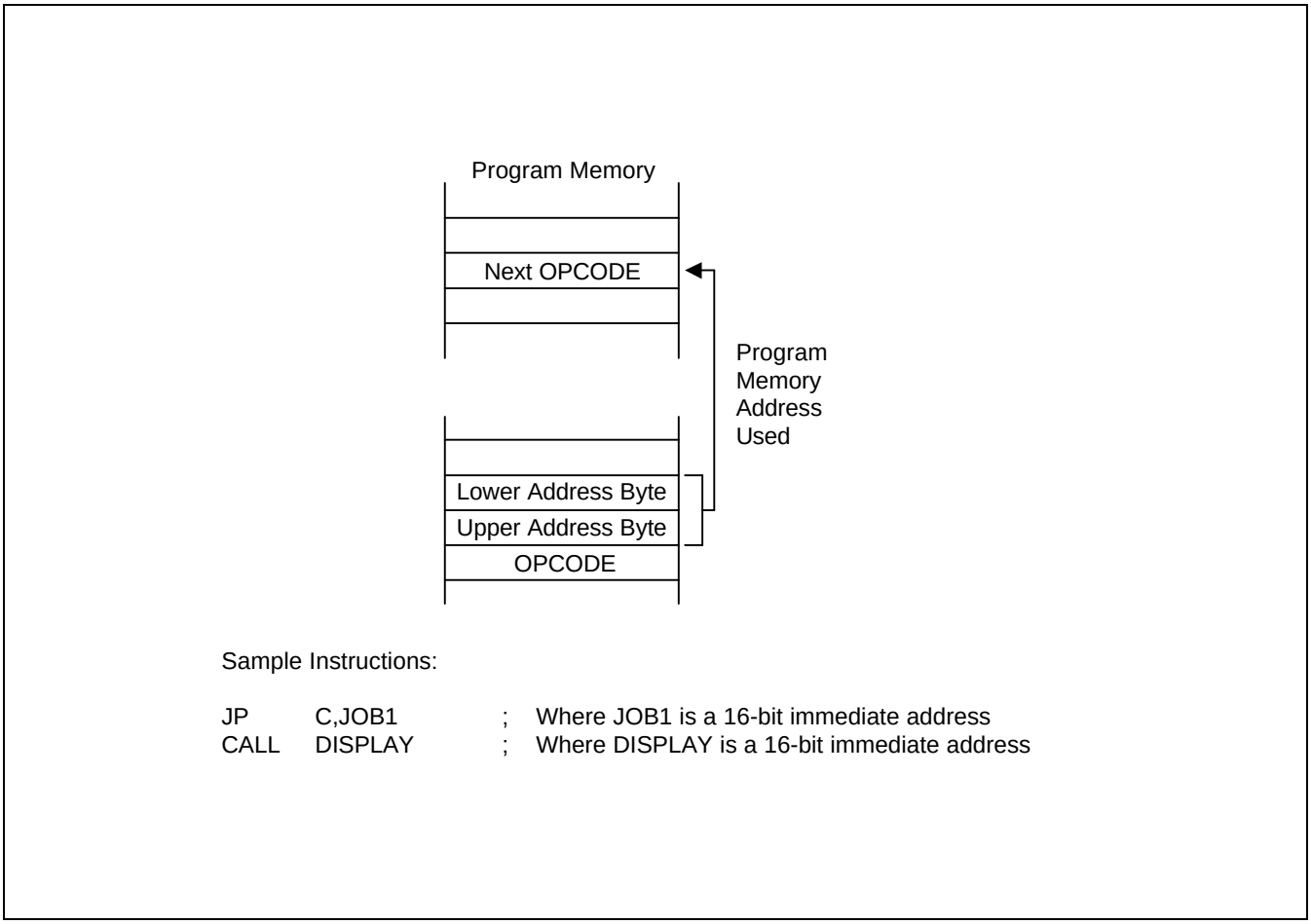



Sample Instructions:

```
LDC     R5,1234H        ;   The values in the program address (1234H)
                            are loaded into register R5.
LDE     R5,1234H        ;   Identical operation to LDC example, except that
                            external program memory is accessed.
```

**Figure 3-10. Direct Addressing for Load Instructions**

## DIRECT ADDRESS MODE (Continued)




Sample Instructions:

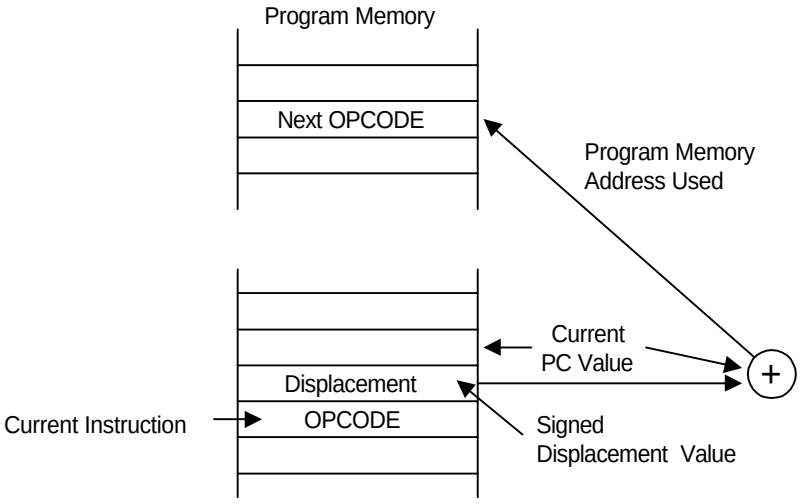
```
JP      C,JOB1      ;   Where JOB1 is a 16-bit immediate address
CALL    DISPLAY     ;   Where DISPLAY is a 16-bit immediate address
```

**Figure 3-11. Direct Addressing for Call and Jump Instructions**

## RELATIVE ADDRESS MODE (RA)

In Relative Address (RA) mode, a two's-complement signed displacement between – 128 and + 127 is specified in the instruction. The displacement value is then added to the current PC value. The result is the address of the next instruction to be executed. Before this addition occurs, the PC contains the address of the instruction immediately following the current instruction.

The instructions that support RA addressing is JR.



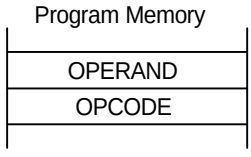

Sample Instructions:

```
JR      ULT,$ + OFFSET      ;   Where OFFSET is a value in the range + 127 to - 128
```

**Figure 3-12. Relative Addressing**

## IMMEDIATE MODE (IM)

In Immediate (IM) addressing mode, the operand value used in the instruction is the value supplied in the operand field itself. Immediate addressing mode is useful for loading constant values into registers.




(The Operand value is in the instruction)

Sample Instruction:

```
LD      R0,#0AAH
```

**Figure 3-13. Immediate Addressing**

# 4 CONTROL REGISTERS

## OVERVIEW

In this section, detailed descriptions of the S3C9654/C9658/P9658 control registers are presented in an easy-to-read format. These descriptions will help familiarize you with the mapped locations in the register file. You can also use them as a quick-reference source when writing application programs.

System and peripheral registers are summarized in Table 4-1. Figure 4-1 illustrates the important features of the standard register description format.

Control register descriptions are arranged in alphabetical order according to register mnemonic. More information about control registers is presented in the context of the various peripheral hardware descriptions in Part II of this manual.

**Table 4-1. System and Peripheral Control Registers**

| Register Name | Mnemonic | Hex | R/W |
|---|---|---|---|
| General purpose register file & Stack area | – | 00-BFH | R/W |
| Working register area | – | C0H-CFH | R/W |
| Timer 0 Counter register | T0CNT | D0H | R |
| Timer 0 data register | T0DATA | D1H | R/W |
| Timer 0 Control register | T0CON | D2H | R/W |
| Location D3H is not mapped. | | | |
| Clock control register | CLKCON | D4H | R/W |
| System FLAG register | FLAGS | D5H | R/W |
| Locations D6H–D8H are not mapped. | | | |
| Stack pointer | SP | D9H | R/W |
| Locations DAH–DBH are not mapped. | | | |
| Basic timer control register | BTCON | DCH | R/W |
| Basic timer counter | BTCNT | DDH | R |
| Location DEH is not mapped. | | | |
| System mode register | SYM | DFH | R/W |
| Port 0 data register | P0 | E0H | R/W |
| Port 1 data register | P1 | E1H | R/W |
| Port 2 data register | P2 | E2H | R/W |
| Port 1 pull-down control | PDCON | E3H | R/W |
| Comparator control mode register | CCON | E4H | R/W |
| Comparison result register | CDATA | E5H | R |
| Port 0 low nibble control register | P0CONL | E6H | R/W |
| Port 0 high nibble control register | P0CONH | E7H | R/W |
| Port 1 high nibble control register | P1CONH | E8H | R/W |
| Port 1 low nibble control register | P1CONL | E9H | R/W |
| Port 0 interrupt control register | P0INT | EAH | R/W |
| Port 0 interrupt pending register | P0PND | EBH | R/W |
| Port 1 interrupt control register | P1INT | ECH | R/W |
| Port 1 interrupt pending register | P1PND | EDH | R/W |
| Port 2 control/interrupt control and pending register | P2CONINT | EEH | R/W |
| Sub oscillator control register | SUBCON | EFH | R/W |

**Table 4-1. System and Peripheral Control Registers (Continued)**

| Register Name | Mnemonic | Hex | R/W |
|---|---|---|---|
| USB function address register | FADDR | F0H | R/W |
| Control endpoint status register | EP0CSR | F1H | R/W |
| Interrupt endpoint status register | EP1CSR | F2H | R/W |
| Control endpoint byte count register | EP0BCNT | F3H | R |
| Control endpoint FIFO register | EP0FIFO | F4H | W |
| Interrupt endpoint FIFO register | EP1FIFO | F5H | W |
| USB interrupt pending register | USBPND | F6H | R/W |
| USB interrupt enable register | USBINT | F7H | R/W |
| USB power management register | PWRMGR | F8H | R/W |
| Locations F9H is not mapped | | | |
| Locations FAH is not mapped | | | |
| USB mode select register | USBSEL | FBH | R/W |
| Locations FCH is not mapped | | | |
| Sink current control register | SNKCON | FDH | R/W |
| USB signal control | XCON | FEH | R/W |
| USB reset register | USBRST | FFH | R/W |

**NOTES:**

1. RESET = value notation
2. "_" = Not used.
3. "x" = Undetermind value.

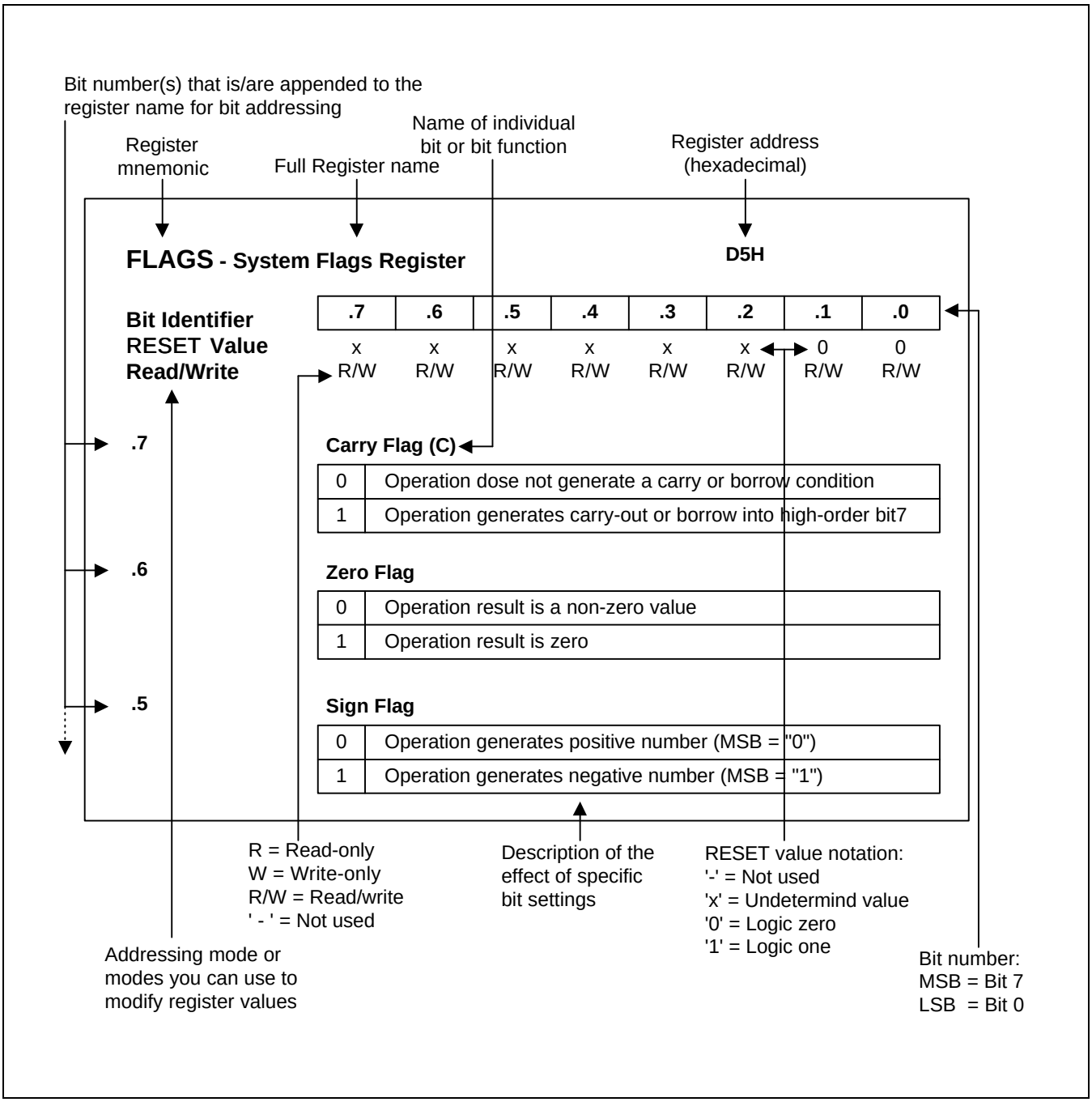



**Figure 4-1. Register Description Format**

# BTCON — Basic Timer Control Register DCH

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**.7 – .4 Watchdog Timer Enable Bits**

| | | | | |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | Disable watchdog function |
| Any other value | | | | Enable watchdog function |

**.3 – .2 Basic Timer Input Clock Selection Bits**

| | | |
|---|---|---|
| 0 | 0 | $f_{OSC}/4096$ |
| 0 | 1 | $f_{OSC}/1024$ |
| 1 | 0 | $f_{OSC}/128$ |
| 1 | 1 | Non divided ($f_{OSC}$) |

**.1 Basic Timer Counter Clear Bit** (note)

| | |
|---|---|
| 0 | No effect |
| 1 | Clear BTCNT |

**.0 Basic Timer Divider Clear Bit** (note)

| | |
|---|---|
| 0 | No effect |
| 1 | Clear both dividers |

**NOTE:** When you write a "1" to BTCON.0 (or BTCON.1), the basic timer counter (or basic timer divider) is cleared. The bit is then cleared automatically to "0".

# CCON — Comparator Mode Register E4H

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**.7 Comparator Enable Bit**

| | |
|---|---|
| 0 | Disable comparator |
| 1 | Enable comparator |

**.6 Conversion Time**

| | |
|---|---|
| 0 | Conversion time (6 × 192/fx) |
| 1 | Conversion time (4 × 12/fx) |

**.5 External Reference Voltage**

| | |
|---|---|
| 0 | Internal reference voltage |
| 1 | External reference voltage |

**.4 – .0 Reference voltage (Vref) selection**

| |
|---|
| $V_{DD} \times (n + 0.5)/24$, n = 0 to 7 |
| $V_{DD} \times (0.3125 + (n – 7)/48)$, n = 8 to 23 |
| $V_{DD} \times (0.6458 + (n – 23)/24)$, n = 24 to 31 |

# CLKCON — System Clock Control Register D4H

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | 0 | – | – | 0 | 0 | – | – | – |
| Read/Write | R/W | – | – | R/W | R/W | – | – | – |

**.7 Oscillator IRQ Wake-up Function Bit**

| | |
|---|---|
| 0 | Enable IRQ for main system oscillator wake-up in power down mode |
| 1 | Disable IRQ for main system oscillator wake-up in power down mode |

**.6 and .5**

Not used for S3C9654/C9658/P9658

**.4 and .3 CPU Clock (System Clock) Selection Bits**

| | | |
|---|---|---|
| 0 | 0 | Divide by 16 ($f_{OSC}$/16) |
| 0 | 1 | Divide by 8 ($f_{OSC}$/8) |
| 1 | 0 | Divide by 2 ($f_{OSC}$/2) |
| 1 | 1 | Non-divided clock ($f_{OSC}$) |

**.2 – .0**

Not used for S3C9654/C9658/P9658

# EP0CSR — Control Endpoint Status Register F1H

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**.7 SETUP_END Clear Bit**

| | |
|---|---|
| 0 | No effect (when write) |
| 1 | Clear SETUP_END (bit4) bit |

**.6 OUT_PKT_RDY Clear Bit**

| | |
|---|---|
| 0 | No effect (when write) |
| 1 | Clear OUT_PKT_RDY (bit0) bit |

**.5 STALL Signal Sending Bit**

| | |
|---|---|
| 0 | No effect (when write) |
| 1 | Send STALL signal to host |

**.4 Setup Transfer End Bit**

| | |
|---|---|
| 0 | No effect (when write) |
| 1 | SIE sets this bit when a control transfer ends before DATA_END (bit3) is set |

**.3 Setup Data End Bit**

| | |
|---|---|
| 0 | No effect (when write) |
| 1 | MCU set this bit after loading or unloading the last packet data into the FIFO |

**.2 STALL Signal Receive Bit**

| | |
|---|---|
| 0 | MCU clear this bit to end the STALL condition |
| 1 | SIE sets this bit if a control transaction is ended due to a protocol violation |

**.1 In Packet Ready Bit**

| | |
|---|---|
| 0 | SIE clear this bit once the packet has been successfully sent to the host |
| 1 | MCU sets this bit after writing a packet of data into Endpoint0 FIFO |

**.0 Out Packet Ready Bit**

| | |
|---|---|
| 0 | No effect (when write) |
| 1 | SIE sets this bit once a valid token is written to the FIFO |

# EP1CSR — Interrupt Endpoint Status Register F2H

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**.7 DATA_TOGGLE Clear Bit**

| | |
|---|---|
| 0 | No effect (when write) |
| 1 | Clears the data toggle sequence bit |

**.6 – .3 Maximum Packet Size Bits**

| | |
|---|---|
| 0 | No effect (when write) |
| 1 | Indicates the maximum packet size for interrupt endpoint |

**.2 FIFO Flush Bit**

| | |
|---|---|
| 0 | No effect (when write) |
| 1 | FIFO is flushed, and IN_PKT_RDY cleared |

**.1 Force STALL Bit**

| | |
|---|---|
| 0 | MCU clears this bit to end the STALL condition |
| 1 | Issues a STALL handshake to USB |

**.0 In Packet Ready Bit**

| | |
|---|---|
| 0 | SIE clear this bit once the packet has been successfully sent to the host |
| 1 | MCU sets this bit after writing a packet of data into Endpoint1 FIFO |

# FLAGS — System Flags Register D5H

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | 0 | 0 | 0 | 0 | – | – | – | – |
| Read/Write | R/W | R/W | R/W | R/W | – | – | – | – |

**.7 Carry Flag (C)**

| | |
|---|---|
| 0 | Operation does not generate a carry or borrow condition |

**.6 Zero Flag (Z)**

| | |
|---|---|
| 0 | Operation result is a non-zero value |
| 1 | Operation result is zero |

**.5 Sign Flag (S)**

| | |
|---|---|
| 0 | Operation generates a positive number (MSB = "0") |
| 1 | Operation generates a negative number (MSB = "1") |

**.4 Overflow Flag (V)**

| | |
|---|---|
| 0 | Operation result is $\leq$ +127 or $\geq$ –128 |
| 1 | Operation result is $\geq$ +127 or $\leq$ –128 |

**.3 – .0**

Not used for S3C9654/C9658/P9658

# P0CONH — Port 0 Control Register (High Byte) (E7H, R/W)

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | – | – | – | 0 | 0 | 0 | 0 |
| Read/Write | – | – | – | – | R/W | R/W | R/W | R/W |

**.7 – .4**

Not used for S3C9654/C9658/P9658

**.3 and .2 Port 0.5 Configuration Control Bits**

| | | |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge external interrupt with pull-up resistor |
| 1 | 0 | Output mode, push-pull |
| 1 | 1 | Not used |

**.1 and .0 Port 0.4 Configuration Control Bits**

| | | |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge external interrupt with pull-up resistor |
| 1 | 0 | Output mode, push-pull |
| 1 | 1 | Not used |

# P0CONL — Port 0 Control Register (Low Byte) (E6H, R/W)

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**.7 and .6** **Port 0.3 Configuration Control Bits**

| | | |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge external interrupt with pull-up resistor |
| 1 | 0 | Output mode, push-pull |
| 1 | 1 | Not used |

**.5 and .4** **Port 0.2 Configuration Control Bits**

| | | |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge external interrupt with pull-up resistor |
| 1 | 0 | Output mode, push-pull |
| 1 | 1 | Not used |

**.3 and .2** **Port 0.1 Configuration Control Bits**

| | | |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge external interrupt with pull-up resistor |
| 1 | 0 | Output mode, push-pull |
| 1 | 1 | Not used |

**.1 and .0** **Port 0.0 Configuration Control Bits**

| | | |
|---|---|---|
| 0 | 0 | Input, rising edge external interrupt. |
| 0 | 1 | Input, falling edge external interrupt with pull-up resistor |
| 1 | 0 | Output mode, n-ch open drain |
| 1 | 1 | Not used |

## P0INT — Port 0 Interrupt Control Register (EAH, R/W)

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | – | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | – | – | R/W | R/W | R/W | R/W | R/W | R/W |

**.7 and .6**

| Not used for S3C9654/C9658/P9658 |
|---|

**.5 – .0** P0.5-P0.0 Interrupt Enable Bits

| | |
|---|---|
| 0 | External interrupt disable |
| 1 | External interrupt enable |

## P0PND — Port 0 Interrupt Pending Register (EBH, R/W)

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | – | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | – | – | R/W | R/W | R/W | R/W | R/W | R/W |

**.7 and .6**

| Not used for S3C9654/C9658/P9658 |
|---|

**.5 – .0** **P0.5-P0.0 Interrupt Pending Bit**

| | |
|---|---|
| 0 | No pending (when read)/clear pending bit (when write) |
| 1 | Pending (when read)/no effect (when write) |

# P1CONH — Port 1 Control Register (High Byte) (E8H, R/W)

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | – | – | – | 0 | 0 | 0 | 0 |
| Read/Write | – | – | – | – | R/W | R/W | R/W | R/W |

**.7 – .4**

Not used for S3C9654/C9658/P9658

**.3 and .2** **Port 1.5 Configuration Control Bits**

| | | |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge external interrupt with pull-up resistor |
| 1 | 0 | Output mode, push-pull |
| 1 | 1 | Comparator input, analog input, (external reference voltage input) |

**.1 and .0** **Port 1.4 Configuration Control Bits**

| | | |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge external interrupt with pull-up resistor |
| 1 | 0 | Output mode, push-pull |
| 1 | 1 | Comparator input, analog input |

# P1CONL — Port 1 Control Register (Low Byte) E9H

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**.7 and .6** **Port 1.3 Configuration Control Bits**

| | | |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge external interrupt with pull-up resistor |
| 1 | 0 | Output mode, push-pull |
| 1 | 1 | Comparator input, analog input |

**.5 and .4** **Port 1.2 Configuration Control Bits**

| | | |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge external interrupt with pull-up resistor |
| 1 | 0 | Output mode, push-pull |
| 1 | 1 | Comparator input, analog input |

**.3 and .2** **Port 1.1 Configuration Control Bits**

| | | |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge external interrupt with pull-up resistor |
| 1 | 0 | Output mode, push-pull |
| 1 | 1 | Comparator input, analog input |

**.1 and .0** **Port 1.0 Configuration Control Bits**

| | | |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge external interrupt with pull-up resistor |
| 1 | 0 | Output mode, push-pull |
| 1 | 1 | Comparator input, analog input |

## P1INT — Port 1 Interrupt Control Register (ECH, R/W)

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | – | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | – | – | R/W | R/W | R/W | R/W | R/W | R/W |

**.7 and .6**

| Not used for S3C9654/C9658/P9658 |
|---|

**.5 – .0** **P1.0-P1.5 Interrupt Enable Bit**

| | |
|---|---|
| 0 | External interrupt disable |
| 1 | External interrupt enable |

## P1PND — Port 1 Interrupt Pending Register EDH

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | – | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | – | – | R/W | R/W | R/W | R/W | R/W | R/W |

**.7 and .6**

| Not used for S3C9654/C9658/P9658 |
|---|

**.5 – .0** **P1.7 Interrupt Pending Bit**

| | |
|---|---|
| 0 | No pending (when read)/clear pending bit (when write) |
| 1 | Pending (when read)/no effect (when write) |

# P2CONINT — Port 2 Control/Interrupt Control and Pending Register EEH

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**.7 and .6** **Port 2.1 Configuration Control Bits**

| | | |
|---|---|---|
| 0 | 0 | Shcmitt trigger input, falling edge external interrupt |
| 0 | 1 | Shcmitt trigger input, falling edge external interrupt with pull-up |
| 1 | 0 | N-CH open drain output mode |
| 1 | 1 | N-CH open drain output mode with pull-up |

**.5 and .4** **Port 2.0 Configuration Control Bits**

| | | |
|---|---|---|
| 0 | 0 | Shcmitt trigger input, falling edge external interrupt |
| 0 | 1 | Shcmitt trigger input, falling edge external interrupt with pull-up |
| 1 | 0 | N-CH open drain output mode |
| 1 | 1 | N-CH open drain output mode with pull-up |

**.3 and .2** **P2.1-P2.0 Interrupt Enable Bit**

| | |
|---|---|
| 0 | External interrupt disable |
| 1 | External interrupt enable |

**.1 and .0** **P2.1-P2.0 Interrupt Pending Bit**

| | |
|---|---|
| 0 | No pending (When read)/clear pending bit (When write) |
| 1 | Pending (When read)/No effect (When write) |

# PDCON — Port 1 Pull-down Resistor Control (E3H, R/W)

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | – | – | – | 0 | 0 | 0 | 0 |
| Read/Write | – | – | – | – | R/W | R/W | R/W | R/W |

**.7 – .4** Not used for S3C9654/C9658/P9658

**.3** "1" = Pull-down enable,"0" pull-down disable when P1 comparator input mode.

**.2 – .0** Select pull-down resistor value from 5 kΩ–19 kΩ
2 kΩ/bit weight at $V_{PORT}$ = 2.5 V.
"0×08" = 19 kΩ
"0×0F" = 5 kΩ, when $V_{PORT}$ = 2.5 V

# PWRMGR — USB Power Management Register (F8H)

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**.7 – .2** Always logic zero

**.1** **RESUME Signal Sending Bit**

| | |
|---|---|
| 0 | RESUME signal is ended |
| 1 | While in suspend state, if the MCU wants to initiate a resume, it writes a 1 to this register for 10 ms (maximum of 15 ms), and clears this register. In suspend mode, if this bit is set to “1”, USB generates resume signaling. |

**.0** **SUSPEND Status Bit**

| | |
|---|---|
| 0 | Cleared automatically when MCU writes a zero to RESUME signal sending bit or when function receives resume signal from the host while in suspend mode |
| 1 | This bit is set when SUSPEND interrupt occur |

# SNKCON — Sink Current Control Register (FDH, R/W)

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | – | – | – | – | – | 0 | 0 |
| Read/Write | – | – | – | – | – | – | R/W | R/W |

**.7 – .2** Not used for S3C9654/C9658/P9658

**.1 – .0** Select sink current of the Port 0.0 n-ch open drain.
"0x00" = 30 mA, "0x01" = 40 mA, "0x02" = 50 mA, :"0x03": = 60 mA
when $V_{PORT}$ = 0.4 V

# SUBCON — SUB_Oscillator Control (EFH, R/W)

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | 0 | – | – | 0 | 0 | 0 | 0 |
| Read/Write | – | R/W | – | – | R/W | R/W | R/W | R/W |

**.7** Not used for S3C9654/C9658/P9658

**.6 Sub_Oscillator Interrupt Pending Bit**

| | |
|---|---|
| 0 | No pending (when read)/clear pending (when write) |
| 1 | Pending (when read)/no effect (when write) |

**.5 – .4** Not used for S3C9654/C9658/P9658

**.3 Sub_Oscillator Interrupt Enable Bit**

| | |
|---|---|
| 0 | Sub_oscillator disable, interrupt disable |
| 1 | Sub_oscillator enable, interrupt enable |

**.2 and .0 Sub_Oscillator Counter Input Clock Selection Bits**

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | $f_{OSC}/2048$ |
| 0 | 0 | 1 | $f_{OSC}/3072$ |
| 0 | 1 | 0 | $f_{OSC}/4096$ |
| 0 | 1 | 1 | $f_{OSC}/6144$ |
| 1 | 0 | 0 | $f_{OSC}/8192$ |
| 1 | 0 | 1 | $f_{OSC}/12288$ |
| 1 | 1 | 0 | $f_{OSC}/16384$ |
| 1 | 1 | 1 | $f_{OSC}/24576$ |

**NOTE:** $f_{OSC}$= 130 KHz (Typ.) when $V_{DD}$ = 5.0 V, $T_A$ = 25 °C.

# SYM — System Mode Register (DFH, R/W)

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | – | – | – | 0 | 0 | 0 | 0 |
| Read/Write | – | – | – | – | R/W | R/W | R/W | R/W |

**.7 – .4**

| Not used for S3C9654/C9658/P9658 |
|---|

**.3** **Global Interrupt Enable Bit**

| | |
|---|---|
| 0 | Global interrupt processing disable |
| 1 | Global interrupt processing enable |

**.2 and .0** **Page Select Bit**

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | Page 0 |
| 0 | 0 | 1 | Page 1 (Not allowed in S3C9654/C9658/P9658) |
| 0 | 1 | 0 | Page 2 (Not allowed in S3C9654/C9658/P9658) |
| 0 | 0 | 1 | Page 3 (Not allowed in S3C9654/C9658/P9658) |

# T0CON — Timer 0 Control Register D2H

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

**.7 and .6 T0 Counter Input Clock Selection Bits**

| | | |
|---|---|---|
| 0 | 0 | $f_{OSC}/4096$ |
| 0 | 1 | $f_{OSC}/256$ |
| 1 | 0 | $f_{OSC}/8$ |
| 1 | 1 | Not used for S3C9654/C9658/P9658 |

**.5 and .4 T0 Operating Mode Selection Bits**

| | | |
|---|---|---|
| 0 | 0 | Interval timer mode (The counter is automatically cleared whenever T0DATA value equals to T0CNT value) |
| 0 | 1 | Invalid selection |
| 1 | 0 | |
| 1 | 1 | Overflow mode (OVF interrupt can occur) |

**.3 T0 Counter Clear Bit (T0CLR)**

| | |
|---|---|
| 0 | No effect |
| 1 | Clear T0 counter (when write) |

**.2 T0 Overflow Interrupt Enable Bit (T0OVF)**

| | |
|---|---|
| 0 | Disable T0 overflow interrupt |
| 1 | Enable T0 overflow interrupt |

**.1 T0 Match Interrupt Enable Bit (T0INT)**

| | |
|---|---|
| 0 | Disable T0 match interrupt |
| 1 | Enable T0 match interrupt |

**.0 T0 Interrupt Pending Bit (T0PND)**

| | |
|---|---|
| 0 | No interrupt pending (when read)/Clear this pending bit (when write) |
| 1 | Interrupt is pending(when read)/No effect(when write) |

**NOTE:** When you write a "1" to T0CON.3, the timer 0 counter is cleared. The bit is then cleared automatically to "0".

# USBINT — USB Interrupt Enable Register F7H

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | – | – | – | – | 0 | 1 | 1 |
| Read/Write | – | – | – | – | – | R/W | R/W | R/W |

**.7 – .3**

| |
|---|
| Not used for S3C9654/C9658/P9658 |

**.2 SUSPEND/RESUME Interrupt Enable Bit**

| | |
|---|---|
| 0 | Disable SUSPEND and RESEME interrupt (default) |
| 1 | Enable SUSPEND and RESEME interrupt |

**.1 ENDPOINT1 Interrupt Pending Bit**

| | |
|---|---|
| 0 | Disable ENDPOINT 1 interrupt |
| 1 | Enable ENDPOINT 1 interrupt (default) |

**.0 ENDPOINT0 Interrupt Pending Bit**

| | |
|---|---|
| 0 | Disable ENDPOINT 0 interrupt |
| 1 | Enable ENDPOINT 0 interrupt (default) |

# USBPND — USB Interrupt Pending Register F6H

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | – | – | – | 0 | 0 | 0 | 0 |
| Read/Write | – | – | – | – | R/W | R/W | R/W | R/W |

**.7 – .4**

| Not used for S3C9654/C9658/P9658 |
|---|

**.3 RESUME Interrupt Pending Bit**

| | |
|---|---|
| 0 | No effect (once read, this bit is cleared automatically) |
| 1 | This bit is set, if RESUME signaling is received while in SUSPEND mode |

**.2 SUSPEND Interrupt Pending Bit**

| | |
|---|---|
| 0 | No effect (once read, this bit is cleared automatically) |
| 1 | This bit is set, when suspend signaling is received |

**.1 ENDPOINT1 Interrupt Pending Bit**

| | |
|---|---|
| 0 | No effect (once read, this bit is cleared automatically) |
| 1 | This bit is set, when endpoint1 needs to be serviced |

**.0 ENDPOINT0 Interrupt Pending Bit**

| | |
|---|---|
| 0 | No effect (once read, this bit is cleared automatically) |
| 1 | This bit is set, while endpoint 0 needs to serviced. It is set under the following conditions:<br>— OUT_PKT_RDY is set<br>— IN_PKT_RDY get cleared<br>— SENT_STALL gets set<br>— DATA_END gets cleared<br>— SETUP_END gets set |

# USBRST — USB RESET Register FFH

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | – | – | – | – | – | – | 1 |
| Read/Write | – | – | – | – | – | – | – | R/W |

**.7 – .1** Not used for S3C9654/C9658/P9658

**.0** **USB Reset Signal Receive Bit**

| | |
|---|---|
| 0 | Clear reset signal bit |
| 1 | This bit is set when host send USB reset signal |

# USBSEL — PORT 2 MODE SELECT REGISTER FBH

| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | – | – | – | – | – | – | 0 |
| Read/Write | – | – | – | – | – | – | – | R/W |

**.7 – .1** Not used for S3C9654/C9658/P9658

**.0** "0" = GPIO Port, PS/2 mode, "1" = USB Port USB mode.

# XCON — USB Signal Control Register (FEH, R/W)

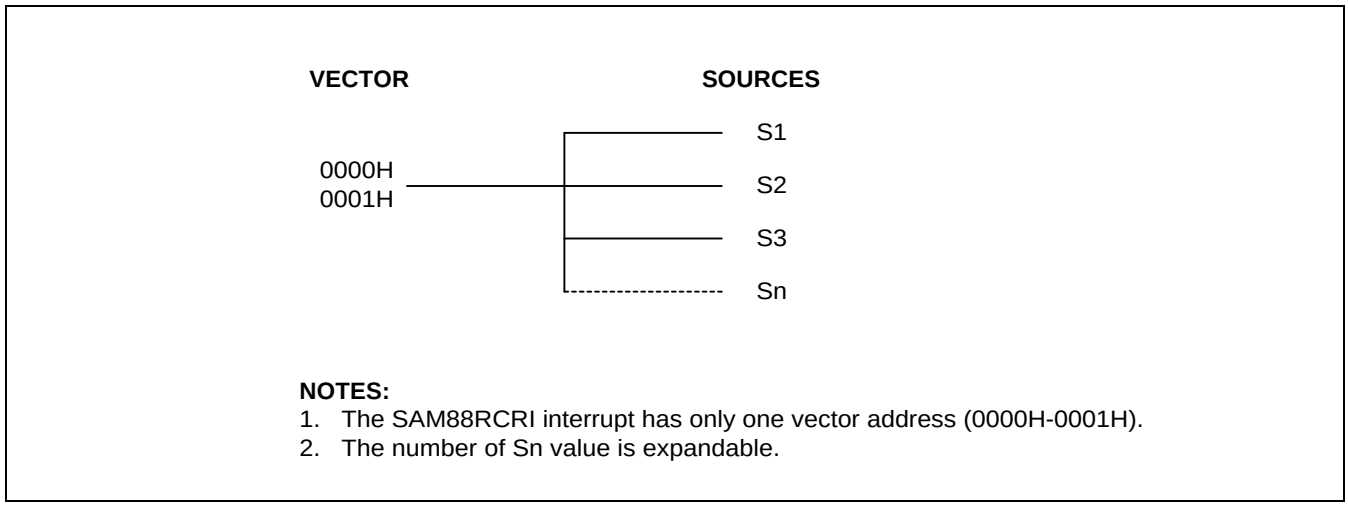
| Bit Identifier | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 |
|---|---|---|---|---|---|---|---|---|
| RESET Value | – | – | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | – | – | R/W | R/W | R/W | R/W | R/W | R/W |

**NOTE:** XCON register value advised by factory (The recommendable value is 1BH).

# 5 INTERRUPT STRUCTURE

## OVERVIEW

The SAM88RCRI interrupt structure has two basic components: a vector, and sources. The number of interrupt sources can be serviced through a interrupt vector which is assigned in ROM address 0000H–0001H.




**NOTES:**
1. The SAM88RCRI interrupt has only one vector address (0000H-0001H).
2. The number of Sn value is expandable.

**Figure 5-1. S3C9-Series Interrupt Type**

### INTERRUPT PROCESSING CONTROL POINTS

Interrupt processing can be controlled in two ways: globally, or by specific interrupt level and source. The system-level control points in the interrupt structure are therefore:

— Global interrupt enable and disable (by EI and DI instructions)

— Interrupt source enable and disable settings in the corresponding peripheral control register(s)

### ENABLE/DISABLE INTERRUPT INSTRUCTIONS (EI, DI)

The system mode register, SYM (DFH), is used to enable and disable interrupt processing.

SYM.3 is the enable and disable bit for global interrupt processing, which you can set by modifying SYM.3. An Enable Interrupt (EI) instruction must be included in the initialization routine that follows a reset operation in order to enable interrupt processing. Although you can manipulate SYM.3 directly to enable and disable interrupts during normal operation, we recommend that you use the EI and DI instructions for this purpose.

## INTERRUPT PENDING FUNCTION TYPES

When the interrupt service routine has executed, the application program's service routine must clear the appropriate pending bit before the return from interrupt subroutine (IRET) occurs.

## INTERRUPT PRIORITY

Because there is not a interrupt priority register in SAM87RI, the order of service is determined by a sequence of source which is executed in interrupt service routine.

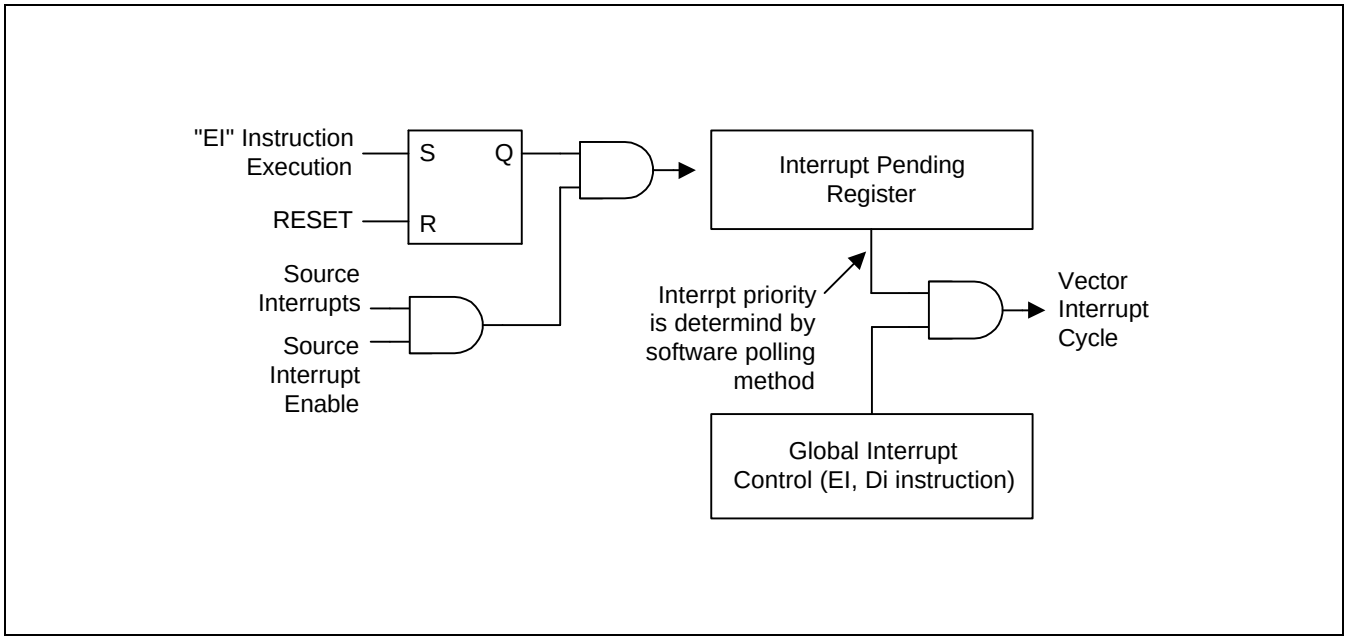



**Figure 5-2. Interrupt Function Diagram**

## INTERRUPT SOURCE SERVICE SEQUENCE

The interrupt request polling and servicing sequence is as follows:

1. A source generates an interrupt request by setting the interrupt request pending bit to "1".
2. The CPU generates an interrupt acknowledge signal.
3. The service routine starts and the source's pending flag is cleared to "0" by software.
4. Interrupt priority must be determined by software polling method.

## INTERRUPT SERVICE ROUTINES

Before an interrupt request can be serviced, the following conditions must be met:

— Interrupt processing must be enabled (EI)
— Interrupt must be enabled at the interrupt's source (peripheral control register)

If all of the above conditions are met, the interrupt request is acknowledged at the end of the instruction cycle. The CPU then initiates an interrupt machine cycle that completes the following processing sequence:

1. Reset (clear to "0") the global interrupt enable bit in the SYM register (DI) to disable all subsequent interrupts.
2. Save the program counter and status flags to stack.
3. Branch to the interrupt vector to fetch the service routine's address.
4. Pass control to the interrupt service routine.

When the interrupt service routine is completed, an Interrupt Return instruction (IRET) occurs. The IRET restores the PC and status flags and sets SYM.3 to "1"(EI), allowing the CPU to process the next interrupt request.

## GENERATING INTERRUPT VECTOR ADDRESSES

The interrupt vector area in the ROM contains the address of the interrupt service routine. Vectored interrupt processing follows this sequence:

1. Push the program counter's low-byte value to stack.
2. Push the program counter's high-byte value to stack.
3. Push the FLAGS register values to stack.
4. Fetch the service routine's high-byte address from the vector address 0000H.
5. Fetch the service routine's low-byte address from the vector address 0001H.
6. Branch to the service routine specified by the 16-bit vector address.

## S3C9654/C9658/P9658 INTERRUPT STRUCTURE

The S3C9654/C9658/P9658 microcontroller has thirteen peripheral interrupt sources:

— Timer 0 match interrupt
— Timer 0 overflow interrupt
— Suspend interrupt
— Resume interrupt
— Two Endpoint interrupts for Endpoint 0 and Endpoint 1
— Three external interrupts for port 0, P0.0–P0.5
— Four external interrupts for port 1, P1.1–P1.5
— Five external interrupts for port 2, P2.0–P2.1 (PS/2 Mode only)
— Internal RC OSC interrupt.

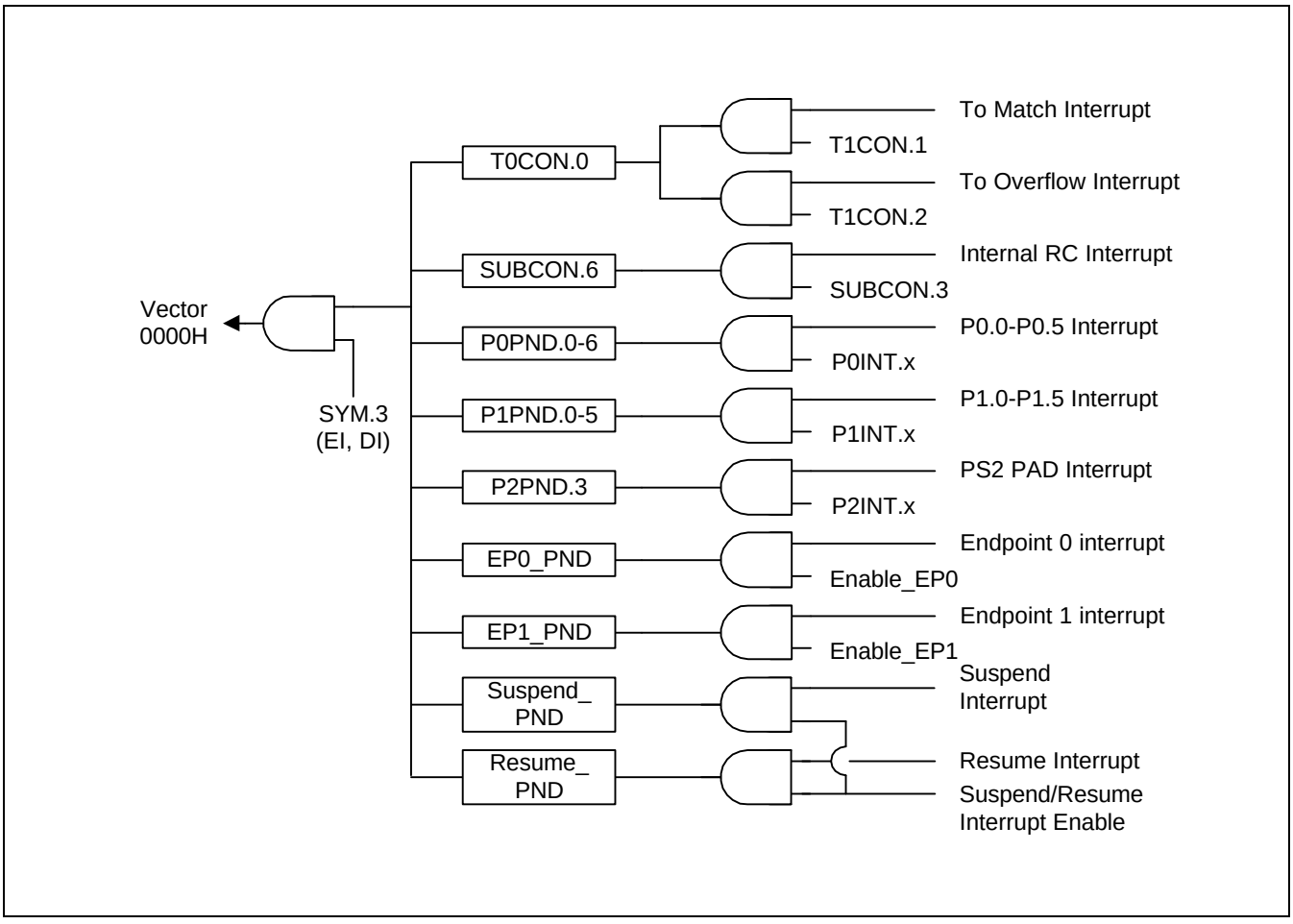



**Figure 5-3. S3C9654/C9658/P9658 Interrupt Structure**

# 6 SAM88RCRI INSTRUCTION SET

## OVERVIEW

The SAM88RCRI instruction set is designed to support the large register file. It includes a full complement of 8-bit arithmetic and logic operations. There are 41 instructions. No special I/O instructions are necessary because I/O control and data registers are mapped directly into the register file. Flexible instructions for bit addressing, rotate, and shift operations complete the powerful data manipulation capabilities of the SAM88RCRI instruction set.

### REGISTER ADDRESSING

To access an individual register, an 8-bit address in the range 0-255 or the 4-bit address of a working register is specified. Paired registers can be used to construct 13-bit program memory or data memory addresses. For detailed information about register addressing, please refer to Section 2, "Address Spaces".

### ADDRESSING MODES

There are six addressing modes: Register (R), Indirect Register (IR), Indexed (X), Direct (DA), Relative (RA), and Immediate (IM). For detailed descriptions of these addressing modes, please refer to Section 3, "Addressing Modes".

**Table 6-1. Instruction Group Summary**

| Mnemonic | Operands | Instruction |
|---|---|---|
| **Load Instructions** | | |
| CLR | dst | Clear |
| LD | dst,src | Load |
| LDC | dst,src | Load program memory |
| LDE | dst,src | Load external data memory |
| LDCD | dst,src | Load program memory and decrement |
| LDED | dst,src | Load external data memory and decrement |
| LDCI | dst,src | Load program memory and increment |
| LDEI | dst,src | Load external data memory and increment |
| POP | dst | Pop from stack |
| PUSH | src | Push to stack |
| **Arithmetic Instructions** | | |
| ADC | dst,src | Add with carry |
| ADD | dst,src | Add |
| CP | dst,src | Compare |
| DEC | dst | Decrement |
| INC | dst | Increment |
| SBC | dst,src | Subtract with carry |
| SUB | dst,src | Subtract |
| **Logic Instructions** | | |
| AND | dst,src | Logical AND |
| COM | dst | Complement |
| OR | dst,src | Logical OR |
| XOR | dst,src | Logical exclusive OR |

**Table 6-1. Instruction Group Summary (Continued)**

| Mnemonic | Operands | Instruction |
|---|---|---|
| **Program Control Instructions** | | |
| CALL | dst | Call procedure |
| IRET | | Interrupt return |
| JP | cc,dst | Jump on condition code |
| JP | dst | Jump unconditional |
| JR | cc,dst | Jump relative on condition code |
| RET | | Return |
| **Bit Manipulation Instructions** | | |
| TCM | dst,src | Test complement under mask |
| TM | dst,src | Test under mask |
| **Rotate and Shift Instructions** | | |
| RL | dst | Rotate left |
| RLC | dst | Rotate left through carry |
| RR | dst | Rotate right |
| RRC | dst | Rotate right through carry |
| SRA | dst | Shift right arithmetic |
| **CPU Control Instructions** | | |
| CCF | | Complement carry flag |
| DI | | Disable interrupts |
| EI | | Enable interrupts |
| IDLE | | Enter Idle mode |
| NOP | | No operation |
| RCF | | Reset carry flag |
| SCF | | Set carry flag |
| STOP | | Enter Stop mode |

## FLAGS REGISTER (FLAGS)

The FLAGS register contains eight bits that describe the current status of CPU operations. Four of these bits, FLAGS.4 – FLAGS.7, can be tested and used with conditional jump instructions;

FLAGS register can be set or reset by instructions as long as its outcome does not affect the flags, such as, Load instruction. Logical and Arithmetic instructions such as, AND, OR, XOR, ADD, and SUB can affect the Flags register. For example, the AND instruction updates the Zero, Sign and Overflow flags based on the outcome of the AND instruction. If the AND instruction uses the Flags register as the destination, then simultaneously, two write will occur to the Flags register producing an unpredictable result.

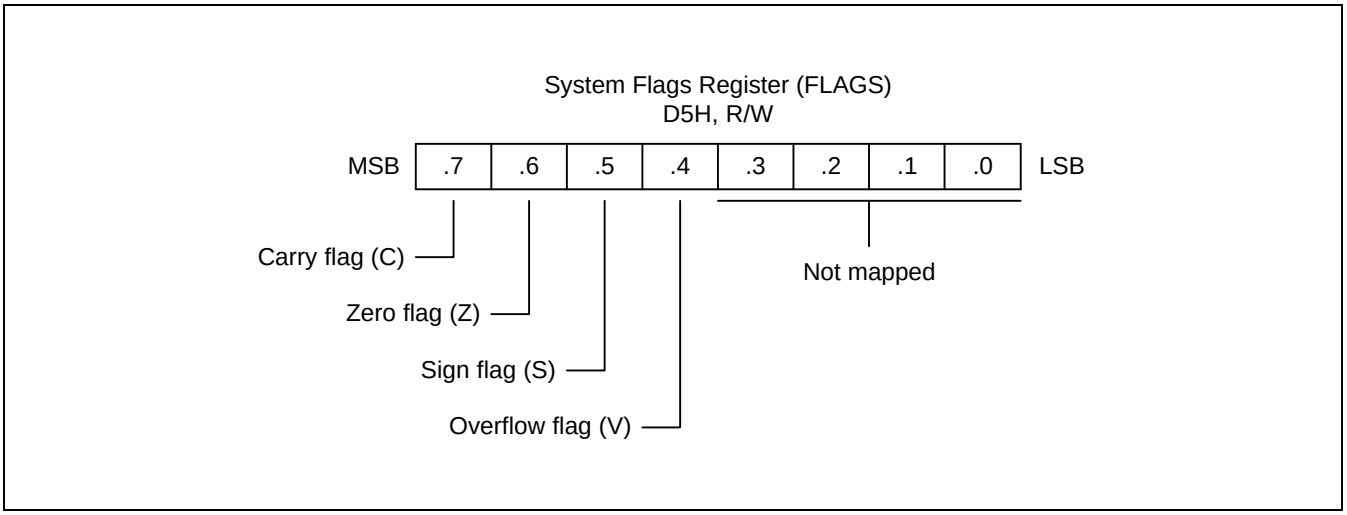



**Figure 6-1. System Flags Register (FLAGS)**

## FLAG DESCRIPTIONS

### Overflow Flag (FLAGS.4, V)

The V flag is set to "1" when the result of a two's-complement operation is greater than + 127 or less than – 128. It is also cleared to "0" following logic operations.

### Sign Flag (FLAGS.5, S)

Following arithmetic, logic, rotate, or shift operations, the sign bit identifies the state of the MSB of the result. A logic zero indicates a positive number and a logic one indicates a negative number.

### Zero Flag (FLAGS.6, Z)

For arithmetic and logic operations, the Z flag is set to "1" if the result of the operation is zero. For operations that test register bits, and for shift and rotate operations, the Z flag is set to "1" if the result is logic zero.

### Carry Flag (FLAGS.7, C)

The C flag is set to "1" if the result from an arithmetic operation generates a carry-out from or a borrow to the bit 7 position (MSB). After rotate and shift operations, it contains the last value shifted out of the specified register. Program instructions can set, clear, or complement the carry flag.

## INSTRUCTION SET NOTATION

**Table 6-2. Flag Notation Conventions**

| Flag | Description |
|---|---|
| C | Carry flag |
| Z | Zero flag |
| S | Sign flag |
| V | Overflow flag |
| 0 | Cleared to logic zero |
| 1 | Set to logic one |
| * | Set or cleared according to operation |
| – | Value is unaffected |
| x | Value is undefined |

**Table 6-3. Instruction Set Symbols**

| Symbol | Description |
|---|---|
| dst | Destination operand |
| src | Source operand |
| @ | Indirect register address prefix |
| PC | Program counter |
| FLAGS | Flags register (D5H) |
| # | Immediate operand or register address prefix |
| H | Hexadecimal number suffix |
| D | Decimal number suffix |
| B | Binary number suffix |
| opc | Opcode |

**Table 6-4. Instruction Notation Conventions**

| Notation | Description | Actual Operand Range |
|---|---|---|
| cc | Condition code | See list of condition codes in Table 6-6. |
| r | Working register only | Rn (n = 0–15) |
| rr | Working register pair | RRp (p = 0, 2, 4, ..., 14) |
| R | Register or working register | reg or Rn (reg = 0–255, n = 0–15) |
| RR | Register pair or working register pair | reg or RRp (reg = 0–254, even number only, where p = 0, 2, ..., 14) |
| Ir | Indirect working register only | @Rn (n = 0–15) |
| IR | Indirect register or indirect working register | @Rn or @reg (reg = 0–255, n = 0–15) |
| Irr | Indirect working register pair only | @RRp (p = 0, 2, ..., 14) |
| IRR | Indirect register pair or indirect working register pair | @RRp or @reg (reg = 0–254, even only, where p = 0, 2, ..., 14) |
| X | Indexed addressing mode | #reg[Rn] (reg = 0–255, n = 0–15) |
| XS | Indexed (short offset) addressing mode | #addr[RRp] (addr = range –128 to +127, where p = 0, 2, ..., 14) |
| xl | Indexed (long offset) addressing mode | #addr [RRp] (addr = range 0–8191, where p = 0, 2, ..., 14) |
| da | Direct addressing mode | addr (addr = range 0–8191) |
| ra | Relative addressing mode | addr (addr = number in the range +127 to –128 that is an offset relative to the address of the next instruction) |
| im | Immediate addressing mode | #data (data = 0–255) |

**Table 6-5. Opcode Quick Reference**

| OPCODE MAP | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | LOWER NIBBLE (HEX) | | | | | | | |
| | – | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| **U** | 0 | DEC R1 | DEC IR1 | ADD r1,r2 | ADD r1,Ir2 | ADD R2,R1 | ADD IR2,R1 | ADD R1,IM | |
| **P** | 1 | RLC R1 | RLC IR1 | ADC r1,r2 | ADC r1,Ir2 | ADC R2,R1 | ADC IR2,R1 | ADC R1,IM | |
| **P** | 2 | INC R1 | INC IR1 | SUB r1,r2 | SUB r1,Ir2 | SUB R2,R1 | SUB IR2,R1 | SUB R1,IM | |
| **E** | 3 | JP IRR1 | | SBC r1,r2 | SBC r1,Ir2 | SBC R2,R1 | SBC IR2,R1 | SBC R1,IM | |
| **R** | 4 | | | OR r1,r2 | OR r1,Ir2 | OR R2,R1 | OR IR2,R1 | OR R1,IM | |
| | 5 | POP R1 | POP IR1 | AND r1,r2 | AND r1,Ir2 | AND R2,R1 | AND IR2,R1 | AND R1,IM | |
| **N** | 6 | COM R1 | COM IR1 | TCM r1,r2 | TCM r1,Ir2 | TCM R2,R1 | TCM IR2,R1 | TCM R1,IM | |
| **I** | 7 | PUSH R2 | PUSH IR2 | TM r1,r2 | TM r1,Ir2 | TM R2,R1 | TM IR2,R1 | TM R1,IM | |
| **B** | 8 | | | | | | | | LD r1, x, r2 |
| **B** | 9 | RL R1 | RL IR1 | | | | | | LD r2, x, r1 |
| **L** | A | | | CP r1,r2 | CP r1,Ir2 | CP R2,R1 | CP IR2,R1 | CP R1,IM | LDC r1, Irr2, xL |
| **E** | B | CLR R1 | CLR IR1 | XOR r1,r2 | XOR r1,Ir2 | XOR R2,R1 | XOR IR2,R1 | XOR R1,IM | LDC r2, Irr2, xL |
| | C | RRC R1 | RRC IR1 | | LDC r1,Irr2 | | | | LD r1, Ir2 |
| **H** | D | SRA R1 | SRA IR1 | | LDC r2,Irr1 | | | LD IR1,IM | LD Ir1, r2 |
| **E** | E | RR R1 | RR IR1 | LDCD r1,Irr2 | LDCI r1,Irr2 | LD R2,R1 | LD R2,IR1 | LD R1,IM | LDC r1, Irr2, xs |
| **X** | F | | | | | CALL IRR1 | LD IR2,R1 | CALL DA1 | LDC r2, Irr1, xs |

**Table 6-5. Opcode Quick Reference (Continued)**

| OPCODE MAP | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | LOWER NIBBLE (HEX) | | | | | | | | |
| | – | 8 | 9 | A | B | C | D | E | F |
| **U** | 0 | LD r1,R2 | LD r2,R1 | | JR cc,RA | LD r1,IM | JP cc,DA | INC r1 | |
| **P** | 1 | ↓ | ↓ | | ↓ | ↓ | ↓ | ↓ | |
| **P** | 2 | | | | | | | | |
| **E** | 3 | | | | | | | | |
| **R** | 4 | | | | | | | | |
| | 5 | | | | | | | | |
| **N** | 6 | | | | | | | | IDLE |
| **I** | 7 | ↓ | ↓ | | ↓ | ↓ | ↓ | ↓ | STOP |
| **B** | 8 | | | | | | | | DI |
| **B** | 9 | | | | | | | | EI |
| **L** | A | | | | | | | | RET |
| **E** | B | | | | | | | | IRET |
| | C | | | | | | | | RCF |
| **H** | D | ↓ | ↓ | | ↓ | ↓ | ↓ | ↓ | SCF |
| **E** | E | | | | | | | | CCF |
| **X** | F | LD r1,R2 | LD r2,R1 | | JR cc,RA | LD r1,IM | JP cc,DA | INC r1 | NOP |

## CONDITION CODES

The opcode of a conditional jump always contains a 4-bit field called the condition code (cc). This specifies under which conditions it is to execute the jump. For example, a conditional jump with the condition code for "equal" after a compare operation only jumps if the two operands are equal. Condition codes are listed in Table 6-6.

The carry (C), zero (Z), sign (S), and overflow (V) flags are used to control the operation of conditional jump instructions.

**Table 6-6. Condition Codes**

| Binary | Mnemonic | Description | Flags Set |
|---|---|---|---|
| 0000 | F | Always false | – |
| 1000 | T | Always true | – |
| 0111 (1) | C | Carry | C = 1 |
| 1111 (1) | NC | No carry | C = 0 |
| 0110 (1) | Z | Zero | Z = 1 |
| 1110 (1) | NZ | Not zero | Z = 0 |
| 1101 | PL | Plus | S = 0 |
| 0101 | MI | Minus | S = 1 |
| 0100 | OV | Overflow | V = 1 |
| 1100 | NOV | No overflow | V = 0 |
| 0110 (1) | EQ | Equal | Z = 1 |
| 1110 (1) | NE | Not equal | Z = 0 |
| 1001 | GE | Greater than or equal | (S XOR V) = 0 |
| 0001 | LT | Less than | (S XOR V) = 1 |
| 1010 | GT | Greater than | (Z OR (S XOR V)) = 0 |
| 0010 | LE | Less than or equal | (Z OR (S XOR V)) = 1 |
| 1111 (1) | UGE | Unsigned greater than or equal | C = 0 |
| 0111 (1) | ULT | Unsigned less than | C = 1 |
| 1011 | UGT | Unsigned greater than | (C = 0 AND Z = 0) = 1 |
| 0011 | ULE | Unsigned less than or equal | (C OR Z) = 1 |

**NOTES:**

1. Indicate condition codes that are related to two different mnemonics but which test the same flag. For example, Z and EQ are both true if the zero flag (Z) is set, but after an ADD instruction, Z would probably be used; after a CP instruction, however, EQ would probably be used.
2. For operations involving unsigned numbers, the special condition codes UGE, ULT, UGT, and ULE must be used.

## INSTRUCTION DESCRIPTIONS

This section contains detailed information and programming examples for each instruction in the SAM88RCRI instruction set. Information is arranged in a consistent format for improved readability and for fast referencing. The following information is included in each instruction description:

— Instruction name (mnemonic)
— Full instruction name
— Source/destination format of the instruction operand
— Shorthand notation of the instruction's operation
— Textual description of the instruction's effect
— Specific flag settings affected by the instruction
— Detailed description of the instruction's format, execution time, and addressing mode(s)
— Programming example(s) explaining how to use the instruction

# ADC — Add With Carry

**ADC** dst,src

**Operation:** dst ¨ dst + src + c

The source operand, along with the setting of the carry flag, is added to the destination operand and the sum is stored in the destination. The contents of the source are unaffected. Two's-complement addition is performed. In multiple precision arithmetic, this instruction permits the carry from the addition of low-order operands to be carried into the addition of high-order operands.

**Flags:**

**C:** Set if there is a carry from the most significant bit of the result; cleared otherwise.
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result is negative; cleared otherwise.
**V:** Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise.
**D:** Always cleared to "0".
**H:** Set if there is a carry from the most significant bit of the low-order four bits of the result; cleared otherwise.

**Format:**

| | | | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | src |
|---|---|---|---|---|---|---|---|
| opc | dst \| src | | 2 | 4 | 12 | r | r |
| | | | | 6 | 13 | r | lr |
| opc | src | dst | 3 | 6 | 14 | R | R |
| | | | | 6 | 15 | R | IR |
| opc | dst | src | 3 | 6 | 16 | R | IM |

**Examples:** Given: R1 = 10H, R2 = 03H, C flag = "1", register 01H = 20H, register 02H = 03H, and register 03H = 0AH:

| | | | |
|---|---|---|---|
| ADC | R1,R2 | → | R1 = 14H, R2 = 03H |
| ADC | R1,@R2 | → | R1 = 1BH, R2 = 03H |
| ADC | 01H,02H | → | Register 01H = 24H, register 02H = 03H |
| ADC | 01H,@02H | → | Register 01H = 2BH, register 02H = 03H |
| ADC | 01H,#11H | → | Register 01H = 32H |

In the first example, destination register R1 contains the value 10H, the carry flag is set to "1", and the source working register R2 contains the value 03H. The statement "ADC R1,R2" adds 03H and the carry flag value ("1") to the destination value 10H, leaving 14H in register R1.

# ADD — Add

**ADD** dst,src

**Operation:** dst ¨ dst + src

The source operand is added to the destination operand and the sum is stored in the destination. The contents of the source are unaffected. Two's-complement addition is performed.

**Flags:**
**C:** Set if there is a carry from the most significant bit of the result; cleared otherwise.
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result is negative; cleared otherwise.
**V:** Set if arithmetic overflow occurred, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise.
**D:** Always cleared to "0".
**H:** Set if a carry from the low-order nibble occurred.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | Addr Mode src |
|---|---|---|---|---|---|
| opc \| dst \| src | 2 | 4 | 02 | r | r |
| | | 6 | 03 | r | lr |
| opc \| src \| dst | 3 | 6 | 04 | R | R |
| | | 6 | 05 | R | IR |
| opc \| dst \| src | 3 | 6 | 06 | R | IM |

**Examples:** Given: R1 = 12H, R2 = 03H, register 01H = 21H, register 02H = 03H, register 03H = 0AH:

| | | | |
|---|---|---|---|
| ADD | R1,R2 | → | R1 = 15H, R2 = 03H |
| ADD | R1,@R2 | → | R1 = 1CH, R2 = 03H |
| ADD | 01H,02H | → | Register 01H = 24H, register 02H = 03H |
| ADD | 01H,@02H | → | Register 01H = 2BH, register 02H = 03H |
| ADD | 01H,#25H | → | Register 01H = 46H |

In the first example, destination working register R1 contains 12H and the source working register R2 contains 03H. The statement "ADD R1,R2" adds 03H to 12H, leaving the value 15H in register R1.

# AND — Logical AND

**AND** dst,src

**Operation:** dst ¨ dst AND src

The source operand is logically ANDed with the destination operand. The result is stored in the destination. The AND operation results in a "1" bit being stored whenever the corresponding bits in the two operands are both logic ones; otherwise a "0" bit value is stored. The contents of the source are unaffected.

**Flags:**
**C:** Unaffected.
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result bit 7 is set; cleared otherwise.
**V:** Always cleared to "0".
**D:** Unaffected.
**H:** Unaffected.

**Format:**

| | | | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | Addr Mode src |
|---|---|---|---|---|---|---|---|
| opc | dst \| src | | 2 | 4 | 52 | r | r |
| | | | | 6 | 53 | r | lr |
| opc | src | dst | 3 | 6 | 54 | R | R |
| | | | | 6 | 55 | R | IR |
| opc | dst | src | 3 | 6 | 56 | R | IM |

**Examples:** Given: R1 = 12H, R2 = 03H, register 01H = 21H, register 02H = 03H, register 03H = 0AH:

| | | | |
|---|---|---|---|
| AND | R1,R2 | → | R1 = 02H, R2 = 03H |
| AND | R1,@R2 | → | R1 = 02H, R2 = 03H |
| AND | 01H,02H | → | Register 01H = 01H, register 02H = 03H |
| AND | 01H,@02H | → | Register 01H = 00H, register 02H = 03H |
| AND | 01H,#25H | → | Register 01H = 21H |

In the first example, destination working register R1 contains the value 12H and the source working register R2 contains 03H. The statement "AND R1,R2" logically ANDs the source operand 03H with the destination operand value 12H, leaving the value 02H in register R1.

# CALL — Call Procedure

**CALL** dst

**Operation:**

SP ¨ SP – 1
@SP ¨ PCL
SP ¨ SP –1
@SP ¨ PCH
PC ¨ dst

The current contents of the program counter are pushed onto the top of the stack. The program counter value used is the address of the first instruction following the CALL instruction. The specified destination address is then loaded into the program counter and points to the first instruction of a procedure. At the end of the procedure the return instruction (RET) can be used to return to the original program flow. RET pops the top of the stack back into the program counter.

**Flags:** No flags are affected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst |
|---|---|---|---|---|
| opc dst | 3 | 14 | F6 | DA |
| opc dst | 2 | 12 | F4 | IRR |

**Examples:** Given: R0 = 15H, R1 = 21H, PC = 1A47H, and SP = 0B2H:

CALL 1521H → SP = 0B0H
(Memory locations 00H = 1AH, 01H = 4AH, where 4AH is the address that follows the instruction.)

CALL @RR0 → SP = 0B0H (00H = 1AH, 01H = 49H)

In the first example, if the program counter value is 1A47H and the stack pointer contains the value 0B2H, the statement "CALL 1521H" pushes the current PC value onto the top of the stack. The stack pointer now points to memory location 00H. The PC is then loaded with the value 1521H, the address of the first instruction in the program sequence to be executed.

If the contents of the program counter and stack pointer are the same as in the first example, the statement "CALL @RR0" produces the same result except that the 49H is stored in stack location 01H (because the two-byte instruction format was used). The PC is then loaded with the value 1521H, the address of the first instruction in the program sequence to be executed.

# CCF — Complement Carry Flag

**CCF**

**Operation:** C ¨ NOT C

The carry flag (C) is complemented. If C = "1", the value of the carry flag is changed to logic zero; if C = "0", the value of the carry flag is changed to logic one.

**Flags:** **C:** Complemented.

No other flags are affected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) |
|---|---|---|---|
| opc | 1 | 4 | EF |

**Example:** Given: The carry flag = "0":

CCF

If the carry flag = "0", the CCF instruction complements it in the FLAGS register (0D5H), changing its value from logic zero to logic one.

# CLR — Clear

**CLR** dst

**Operation:** dst ¨ "0"

The destination location is cleared to "0".

**Flags:** No flags are affected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst |
|---|---|---|---|---|
| opc \| dst | 2 | 4 | B0 | R |
| | | 4 | B1 | IR |

**Examples:** Given: Register 00H = 4FH, register 01H = 02H, and register 02H = 5EH:

```
CLR   00H    →   Register 00H = 00H
CLR   @01H   →   Register 01H = 02H, register 02H = 00H
```

In Register (R) addressing mode, the statement "CLR 00H" clears the destination register 00H value to 00H. In the second example, the statement "CLR @01H" uses Indirect Register (IR) addressing mode to clear the 02H register value to 00H.

# COM — Complement

**COM** dst

**Operation:** dst ¨ NOT dst

The contents of the destination location are complemented (one's complement); all "1s" are changed to "0s", and vice-versa.

**Flags:**
**C:** Unaffected.
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result bit 7 is set; cleared otherwise.
**V:** Always reset to "0".
**D:** Unaffected.
**H:** Unaffected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst |
|---|---|---|---|---|
| opc \| dst | 2 | 4 | 60 | R |
| | | 4 | 61 | IR |

**Examples:** Given: R1 = 07H and register 07H = 0F1H:

```
COM     R1      →     R1 = 0F8H
COM     @R1     →     R1 = 07H, register 07H = 0EH
```

In the first example, destination working register R1 contains the value 07H (00000111B). The statement "COM R1" complements all the bits in R1: all logic ones are changed to logic zeros, and vice-versa, leaving the value 0F8H (11111000B).

In the second example, Indirect Register (IR) addressing mode is used to complement the value of destination register 07H (11110001B), leaving the new value 0EH (00001110B).

# CP — Compare

**CP** dst,src

**Operation:** dst – src

The source operand is compared to (subtracted from) the destination operand, and the appropriate flags are set accordingly. The contents of both operands are unaffected by the comparison.

**Flags:**
**C:** Set if a "borrow" occurred (src > dst); cleared otherwise.
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result is negative; cleared otherwise.
**V:** Set if arithmetic overflow occurred, that is, if the operands were of opposite signs and the sign of the result is of the same as the sign of the source operand; cleared otherwise.
**D:** Unaffected.
**H:** Unaffected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | src |
|---|---|---|---|---|---|
| opc \| dst \| src | 2 | 4 | A2 | r | r |
| | | 6 | A3 | r | lr |
| opc \| src \| dst | 3 | 6 | A4 | R | R |
| | | 6 | A5 | R | IR |
| opc \| dst \| src | 3 | 6 | A6 | R | IM |

**Examples:** 1. Given: R1 = 02H and R2 = 03H:

```
CP      R1,R2      →      Set the C and S flags
```

Destination working register R1 contains the value 02H and source register R2 contains the value 03H. The statement "CP R1,R2" subtracts the R2 value (source/subtrahend) from the R1 value (destination/minuend). Because a "borrow" occurs and the difference is negative, C and S are "1".

2. Given: R1 = 05H and R2 = 0AH:

```
        CP      R1,R2
        JP      UGE,SKIP
        INC     R1
SKIP    LD      R3,R1
```

In this example, destination working register R1 contains the value 05H which is less than the contents of the source working register R2 (0AH). The statement "CP R1,R2" generates C = "1" and the JP instruction does not jump to the SKIP location. After the statement "LD R3,R1" executes, the value 06H remains in working register R3.

# DEC — Decrement

**DEC** dst

**Operation:** dst ¨ dst – 1

The contents of the destination operand are decremented by one.

**Flags:**
**C:** Unaffected.
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if result is negative; cleared otherwise.
**V:** Set if arithmetic overflow occurred, that is, dst value is –128(80H) and result value is +127(7FH); cleared otherwise.
**D:** Unaffected.
**H:** Unaffected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst |
|---|---|---|---|---|
| opc dst | 2 | 4 | 00 | R |
| | | 4 | 01 | IR |

**Examples:** Given: R1 = 03H and register 03H = 10H:

```
DEC   R1    →   R1 = 02H
DEC   @R1   →   Register 03H = 0FH
```

In the first example, if working register R1 contains the value 03H, the statement "DEC R1" decrements the hexadecimal value by one, leaving the value 02H. In the second example, the statement "DEC @R1" decrements the value 10H contained in the destination register 03H by one, leaving the value 0FH.

# DI — Disable Interrupts

**DI**

**Operation:** SYM (2) ¨ 0

Bit zero of the system mode register, SYM.2, is cleared to "0", globally disabling all interrupt processing. Interrupt requests will continue to set their respective interrupt pending bits, but the CPU will not service them while interrupt processing is disabled.

**Flags:** No flags are affected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) |
|---|---|---|---|
| opc | 1 | 4 | 8F |

**Example:** Given: SYM = 04H:

DI

If the value of the SYM register is 04H, the statement "DI" leaves the new value 00H in the register and clears SYM.2 to "0", disabling interrupt processing.

# EI — Enable Interrupts

**EI**

**Operation:** SYM (2) ¨ 1

An EI instruction sets bit 2 of the system mode register, SYM.2 to "1". This allows interrupts to be serviced as they occur. If an interrupt's pending bit was set while interrupt processing was disabled (by executing a DI instruction), it will be serviced when you execute the EI instruction.

**Flags:** No flags are affected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) |
|---|---|---|---|
| opc | 1 | 4 | 9F |

**Example:** Given: SYM = 00H:

EI

If the SYM register contains the value 00H, that is, if interrupts are currently disabled, the statement "EI" sets the SYM register to 04H, enabling all interrupts (SYM.2 is the enable bit for global interrupt processing).

# IDLE — Idle Operation

**IDLE**

**Operation:**

The IDLE instruction stops the CPU clock while allowing system clock oscillation to continue. Idle mode can be released by an interrupt request (IRQ) or an external reset operation.

**Flags:** No flags are affected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | src |
|---|---|---|---|---|---|
| opc | 1 | 4 | 6F | – | – |

**Example:** The instruction

IDLE

stops the CPU clock but not the system clock.

# INC — Increment

**INC** dst

**Operation:** dst ¨ dst + 1

The contents of the destination operand are incremented by one.

**Flags:**
**C:** Unaffected.
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result is negative; cleared otherwise.
**V:** Set if arithmetic overflow occurred, that is dst value is +127(7FH) and result is –128(80H); cleared otherwise.
**D:** Unaffected.
**H:** Unaffected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst |
|---|---|---|---|---|
| dst \| opc | 1 | 4 | rE<br>r = 0 to F | r |
| opc \| dst | 2 | 4 | 20 | R |
| | | 4 | 21 | IR |

**Examples:** Given: R0 = 1BH, register 00H = 0CH, and register 1BH = 0FH:

```
INC     R0      →       R0 = 1CH
INC     00H     →       Register 00H = 0DH
INC     @R0     →       R0 = 1BH, register 01H = 10H
```

In the first example, if destination working register R0 contains the value 1BH, the statement "INC R0" leaves the value 1CH in that same register.

The next example shows the effect an INC instruction has on register 00H, assuming that it contains the value 0CH.

In the third example, INC is used in Indirect Register (IR) addressing mode to increment the value of register 1BH from 0FH to 10H.

# IRET — Interrupt Return

**IRET** IRET

**Operation:**
FLAGS ¨ @SP
SP ¨ SP + 1
PC ¨ @SP
SP ¨ SP + 2
SYM(2) ¨ 1

This instruction is used at the end of an interrupt service routine. It restores the flag register and the program counter. It also re-enables global interrupts.

**Flags:** All flags are restored to their original settings (that is, the settings before the interrupt occurred).

**Format:**

| IRET (Normal) | Bytes | Cycles | Opcode (Hex) |
|---|---|---|---|
| opc | 1 | 6 | BF |

# JP — Jump

**JP** cc,dst (Conditional)

**JP** dst (Unconditional)

**Operation:** If cc is true, PC ¨ dst

The conditional JUMP instruction transfers program control to the destination address if the condition specified by the condition code (cc) is true; otherwise, the instruction following the JP instruction is executed. The unconditional JP simply replaces the contents of the PC with the contents of the specified register pair. Control then passes to the statement addressed by the PC.

**Flags:** No flags are affected.

**Format:** (1)

| (2) | Bytes | Cycles | Opcode (Hex) | Addr Mode dst |
|---|---|---|---|---|
| cc \| opc / dst | 3 | 8 (3) | ccD<br>cc = 0 to F | DA |
| opc / dst | 2 | 8 | 30 | IRR |

**NOTES:**

1. The 3-byte format is used for a conditional jump and the 2-byte format for an unconditional jump.
2. In the first byte of the three-byte instruction format (conditional jump), the condition code and the opcode are both four bits.

**Examples:** Given: The carry flag (C) = "1", register 00 = 01H, and register 01 = 20H:

```
JP   C,LABEL_W  →   LABEL_W = 1000H, PC = 1000H
JP   @00H       →   PC = 0120H
```

The first example shows a conditional JP. Assuming that the carry flag is set to "1", the statement "JP C,LABEL_W" replaces the contents of the PC with the value 1000H and transfers control to that location. Had the carry flag not been set, control would then have passed to the statement immediately following the JP instruction.

The second example shows an unconditional JP. The statement "JP @00" replaces the contents of the PC with the contents of the register pair 00H and 01H, leaving the value 0120H.

# JR — Jump Relative

**JR** cc,dst

**Operation:** If cc is true, PC ¨ PC + dst

If the condition specified by the condition code (cc) is true, the relative address is added to the program counter and control passes to the statement whose address is now in the program counter; otherwise, the instruction following the JR instruction is executed (See list of condition codes).

The range of the relative address is +127, –128, and the original value of the program counter is taken to be the address of the first instruction byte following the JR statement.

**Flags:** No flags are affected.

**Format:**

(1)

| cc \| opc | dst |
|---|---|

| Bytes | Cycles | Opcode (Hex) | Addr Mode dst |
|---|---|---|---|
| 2 | 6 (2) | ccB | RA |
| | | cc = 0 to F | |

**NOTE**: In the first byte of the two-byte instruction format, the condition code and the opcode are each four bits.

**Example:** Given: The carry flag = "1" and LABEL_X = 1FF7H:

JR C,LABEL_X → PC = 1FF7H

If the carry flag is set (that is, if the condition code is true), the statement "JR C,LABEL_X" will pass control to the statement whose address is now in the PC. Otherwise, the program instruction following the JR would be executed.

# LD — Load

**LD** dst,src

**Operation:** dst ¨ src

The contents of the source are loaded into the destination. The source's contents are unaffected.

**Flags:** No flags are affected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | src |
|---|---|---|---|---|---|
| dst \| opc / src | 2 | 4 | rC | r | IM |
| | | 4 | r8 | r | R |
| src \| opc / dst | 2 | 4 | r9 | R | r |
| | | | r = 0 to F | | |
| opc / dst \| src | 2 | 4 | C7 | r | lr |
| | | 4 | D7 | lr | r |
| opc / src / dst | 3 | 6 | E4 | R | R |
| | | 6 | E5 | R | IR |
| opc / dst / src | 3 | 6 | E6 | R | IM |
| | | 6 | D6 | IR | IM |
| opc / src / dst | 3 | 6 | F5 | IR | R |
| opc / dst \| src / x | 3 | 6 | 87 | r | x [r] |
| opc / src \| dst / x | 3 | 6 | 97 | x [r] | r |

# LD — Load

**LD** (Continued)

**Examples:** Given: R0 = 01H, R1 = 0AH, register 00H = 01H, register 01H = 20H, register 02H = 02H, LOOP = 30H, and register 3AH = 0FFH:

| | | | |
|---|---|---|---|
| LD | R0,#10H | → | R0 = 10H |
| LD | R0,01H | → | R0 = 20H, register 01H = 20H |
| LD | 01H,R0 | → | Register 01H = 01H, R0 = 01H |
| LD | R1,@R0 | → | R1 = 20H, R0 = 01H |
| LD | @R0,R1 | → | R0 = 01H, R1 = 0AH, register 01H = 0AH |
| LD | 00H,01H | → | Register 00H = 20H, register 01H = 20H |
| LD | 02H,@00H | → | Register 02H = 20H, register 00H = 01H |
| LD | 00H,#0AH | → | Register 00H = 0AH |
| LD | @00H,#10H | → | Register 00H = 01H, register 01H = 10H |
| LD | @00H,02H | → | Register 00H = 01H, register 01H = 02, register 02H = 02H |
| LD | R0,#LOOP[R1] | → | R0 = 0FFH, R1 = 0AH |
| LD | #LOOP[R0],R1 | → | Register 31H = 0AH, R0 = 01H, R1 = 0AH |

# LDC/LDE — Load Memory

**LDC/LDE** dst,src

**Operation:** dst ¨ src

This instruction loads a byte from program or data memory into a working register or vice-versa. The source values are unaffected. LDC refers to program memory and LDE to data memory. The assembler makes 'Irr' or 'rr' values an even number for program memory and an odd number for data memory.

**Flags:** No flags are affected.

**Format:**

| | | | | | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | Addr Mode src |
|---|---|---|---|---|---|---|---|---|---|
| 1. | opc | dst \| src | | | 2 | 10 | C3 | r | Irr |
| 2. | opc | src \| dst | | | 2 | 10 | D3 | Irr | r |
| 3. | opc | dst \| src | XS | | 3 | 12 | E7 | r | XS [rr] |
| 4. | opc | src \| dst | XS | | 3 | 12 | F7 | XS [rr] | r |
| 5. | opc | dst \| src | $XL_L$ | $XL_H$ | 4 | 14 | A7 | r | XL [rr] |
| 6. | opc | src \| dst | $XL_L$ | $XL_H$ | 4 | 14 | B7 | XL [rr] | r |
| 7. | opc | dst \| 0000 | $DA_L$ | $DA_H$ | 4 | 14 | A7 | r | DA |
| 8. | opc | src \| 0000 | $DA_L$ | $DA_H$ | 4 | 14 | B7 | DA | r |
| 9. | opc | dst \| 0001 | $DA_L$ | $DA_H$ | 4 | 14 | A7 | r | DA |
| 10. | opc | src \| 0001 | $DA_L$ | $DA_H$ | 4 | 14 | B7 | DA | r |

**NOTES:**

1. The source (src) or working register pair [rr] for formats 5 and 6 cannot use register pair 0–1.
2. For formats 3 and 4, the destination address 'XS [rr]' and the source address 'XS [rr]' are each one byte.
3. For formats 5 and 6, the destination address 'XL [rr] and the source address 'XL [rr]' are each two bytes.
4. The DA and r source values for formats 7 and 8 are used to address program memory; the second set of values, used in formats 9 and 10, are used to address data memory.

# LDC/LDE — Load Memory

**LDC/LDE** (Continued)

**Examples:** Given: R0 = 11H, R1 = 34H, R2 = 01H, R3 = 04H, R4 = 00H, R5 = 60H; Program memory locations 0061 = AAH, 0103H = 4FH, 0104H = 1A, 0105H = 6DH, and 1104H = 88H. External data memory locations 0061H = BBH, 0103H = 5FH, 0104H = 2AH, 0105H = 7DH, and 1104H = 98H:

```
LDC     R0,@RR2          ; R0 ¨ contents of program memory location 0104H
                         ; R0 = 1AH, R2 = 01H, R3 = 04H
LDE     R0,@RR2          ; R0 ¨ contents of external data memory location 0104H
                         ; R0 = 2AH, R2 = 01H, R3 = 04H
LDC *   @RR2,R0          ; 11H (contents of R0) is loaded into program memory
                         ; location 0104H (RR2),
                         ; working registers R0, R2, R3 Æ no change
LDE     @RR2,R0          ; 11H (contents of R0) is loaded into external data memory
                         ; location 0104H (RR2),
                         ; working registers R0, R2, R3 Æ no change
LDC     R0,#01H[RR4]     ; R0 ¨ contents of program memory location 0061H
                         ; (01H + RR4),
                         ; R0 = AAH, R2 = 00H, R3 = 60H
LDE     R0,#01H[RR4]     ; R0 ¨ contents of external data memory location 0061H
                         ; (01H + RR4), R0 = BBH, R4 = 00H, R5 = 60H
LDC (note)  #01H[RR4],R0 ; 11H (contents of R0) is loaded into program memory
                         ; location 0061H (01H + 0060H)
LDE     #01H[RR4],R0     ; 11H (contents of R0) is loaded into external data memory
                         ; location 0061H (01H + 0060H)
LDC     R0,#1000H[RR2]   ; R0 ¨ contents of program memory location 1104H
                         ; (1000H + 0104H), R0 = 88H, R2 = 01H, R3 = 04H
LDE     R0,#1000H[RR2]   ; R0 ¨ contents of external data memory location 1104H
                         ; (1000H + 0104H), R0 = 98H, R2 = 01H, R3 = 04H
LDC     R0,1104H         ; R0 ¨ contents of program memory location 1104H,
                         ; R0 = 88H
LDE     R0,1104H         ; R0 ¨ contents of external data memory location 1104H,
                         ; R0 = 98H
LDC (note)  1105H,R0     ; 11H (contents of R0) is loaded into program memory
                         ; location 1105H, (1105H) ¨ 11H
LDE     1105H,R0         ; 11H (contents of R0) is loaded into external data memory
                         ; location 1105H, (1105H) ¨ 11H
```

**NOTE:** These instructions are not supported by masked ROM type devices.

# LDCD/LDED — Load Memory and Decrement

**LDCD/LDED** dst,src

**Operation:** dst ¨ src

rr ¨ rr – 1

These instructions are used for user stacks or block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair. The contents of the source location are loaded into the destination location. The memory address is then decremented. The contents of the source are unaffected.

LDCD references program memory and LDED references external data memory. The assembler makes 'Irr' an even number for program memory and an odd number for data memory.

**Flags:** No flags are affected.

**Format:**

| | | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | src |
|---|---|---|---|---|---|---|
| opc | dst \| src | 2 | 10 | E2 | r | Irr |

**Examples:** Given: R6 = 10H, R7 = 33H, R8 = 12H, program memory location 1033H = 0CDH, and external data memory location 1033H = 0DDH:

```
LDCD    R8,@RR6     ; 0CDH (contents of program memory location 1033H) is
                    ; loaded into R8 and RR6 is decremented by one
                    ; R8 = 0CDH, R6 = 10H, R7 = 32H (RR6 ← RR6 - 1)
LDED    R8,@RR6     ; 0DDH (contents of data memory location 1033H) is
                    ; loaded into R8 and RR6 is decremented by one
                    ; (RR6 ← RR6 - 1) R8 = 0DDH, R6 = 10H, R7 = 32H
```

# LDCI/LDEI — Load Memory and Increment

**LDCI/LDEI** dst,src

**Operation:** dst ¨ src

rr ¨ rr + 1

These instructions are used for user stacks or block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair. The contents of the source location are loaded into the destination location. The memory address is then incremented automatically. The contents of the source are unaffected.

LDCI refers to program memory and LDEI refers to external data memory. The assembler makes 'Irr' even for program memory and odd for data memory.

**Flags:** No flags are affected.

**Format:**

| | | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | src |
|---|---|---|---|---|---|---|
| opc | dst \| src | 2 | 10 | E3 | r | Irr |

**Examples:** Given: R6 = 10H, R7 = 33H, R8 = 12H, program memory locations 1033H = 0CDH and 1034H = 0C5H; external data memory locations 1033H = 0DDH and 1034H = 0D5H:

```
LDCI    R8,@RR6     ; 0CDH (contents of program memory location 1033H) is
                    ; loaded into R8 and RR6 is incremented by one
                    ; (RR6 ¨ RR6 + 1) R8 = 0CDH, R6 = 10H, R7 = 34H
LDEI    R8,@RR6     ; 0DDH (contents of data memory location 1033H) is
                    ; loaded into R8 and RR6 is incremented by one
                    ; (RR6 ¨ RR6 + 1) R8 = 0DDH, R6 = 10H, R7 = 34H
```

# NOP — No Operation

**NOP**

**Operation:** No action is performed when the CPU executes this instruction. Typically, one or more NOPs are executed in sequence in order to effect a timing delay of variable duration.

**Flags:** No flags are affected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) |
|---|---|---|---|
| opc | 1 | 4 | FF |

**Example:** When the instruction

NOP

is encountered in a program, no operation occurs. Instead, there is a delay in instruction execution time.

# OR — Logical OR

**OR** dst,src

**Operation:** dst ¨ dst OR src

The source operand is logically ORed with the destination operand and the result is stored in the destination. The contents of the source are unaffected. The OR operation results in a "1" being stored whenever either of the corresponding bits in the two operands is a "1"; otherwise a "0" is stored.

**Flags:**
**C:** Unaffected.
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result bit 7 is set; cleared otherwise.
**V:** Always cleared to "0".
**D:** Unaffected.
**H:** Unaffected.

**Format:**

| | | | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | Addr Mode src |
|---|---|---|---|---|---|---|---|
| opc | dst \| src | | 2 | 4 | 42 | r | r |
| | | | | 6 | 43 | r | lr |
| opc | src | dst | 3 | 6 | 44 | R | R |
| | | | | 6 | 45 | R | IR |
| opc | dst | src | 3 | 6 | 46 | R | IM |

**Examples:** Given: R0 = 15H, R1 = 2AH, R2 = 01H, register 00H = 08H, register 01H = 37H, and register 08H = 8AH:

| | | | |
|---|---|---|---|
| OR | R0,R1 | → | R0 = 3FH, R1 = 2AH |
| OR | R0,@R2 | → | R0 = 37H, R2 = 01H, register 01H = 37H |
| OR | 00H,01H | → | Register 00H = 3FH, register 01H = 37H |
| OR | 01H,@00H | → | Register 00H = 08H, register 01H = 0BFH |
| OR | 00H,#02H | → | Register 00H = 0AH |

In the first example, if working register R0 contains the value 15H and register R1 the value 2AH, the statement "OR R0,R1" logical-ORs the R0 and R1 register contents and stores the result (3FH) in destination register R0.

The other examples show the use of the logical OR instruction with the various addressing modes and formats.

# POP — Pop From Stack

**POP** dst

**Operation:** dst ¨ @SP

SP ¨ SP + 1

The contents of the location addressed by the stack pointer are loaded into the destination. The stack pointer is then incremented by one.

**Flags:** No flags affected.

**Format:**

| | | Bytes | Cycles | Opcode (Hex) | Addr Mode dst |
|---|---|---|---|---|---|
| opc | dst | 2 | 8 | 50 | R |
| | | | 8 | 51 | IR |

**Examples:** Given: Register 00H = 01H, register 01H = 1BH, SP (0D9H) = 0BBH, and stack register 0BBH = 55H:

POP 00H → Register 00H = 55H, SP = 0BCH
POP @00H → Register 00H = 01H, register 01H = 55H, SP = 0BCH

In the first example, general register 00H contains the value 01H. The statement "POP 00H" loads the contents of location 0BBH (55H) into destination register 00H and then increments the stack pointer by one. Register 00H then contains the value 55H and the SP points to location 0BCH.

# PUSH — Push To Stack

**PUSH** src

**Operation:** SP ¨ SP – 1

@SP ¨ src

A PUSH instruction decrements the stack pointer value and loads the contents of the source (src) into the location addressed by the decremented stack pointer. The operation then adds the new value to the top of the stack.

**Flags:** No flags are affected.

**Format:**

| | | Bytes | Cycles | Opcode (Hex) | Addr Mode dst |
|---|---|---|---|---|---|
| opc | src | 2 | 8 | 70 | R |
| | | | 8 | 71 | IR |

**Examples:** Given: Register 40H = 4FH, register 4FH = 0AAH, SP = 0C0H:

PUSH 40H → Register 40H = 4FH, stack register 0BFH = 4FH, SP = 0BFH

PUSH @40H → Register 40H = 4FH, register 4FH = 0AAH, stack register 0BFH = 0AAH, SP = 0BFH

In the first example, if the stack pointer contains the value 0C0H, and general register 40H the value 4FH, the statement "PUSH 40H" decrements the stack pointer from 0C0 to 0BFH. It then loads the contents of register 40H into location 0BFH. Register 0BFH then contains the value 4FH and SP points to location 0BFH.

# RCF — Reset Carry Flag

**RCF** RCF

**Operation:** C ¨ 0

The carry flag is cleared to logic zero, regardless of its previous value.

**Flags:** **C:** Cleared to "0".

No other flags are affected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) |
|---|---|---|---|
| opc | 1 | 4 | CF |

**Example:** Given: C = "1" or "0":

The instruction RCF clears the carry flag (C) to logic zero.

# RET — Return

**RET**

**Operation:** PC ¨ @SP

SP ¨ SP + 2

The RET instruction is normally used to return to the previously executing procedure at the end of a procedure entered by a CALL instruction. The contents of the location addressed by the stack pointer are popped into the program counter. The next statement that is executed is the one that is addressed by the new program counter value.

**Flags:** No flags are affected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) |
|---|---|---|---|
| opc | 1 | 8 | AF |

**Example:** Given: SP = 0BCH, (SP) = 101AH, and PC = 1234:

RET → PC = 101AH, SP = 0BEH

The statement "RET" pops the contents of stack pointer location 0BCH (10H) into the high byte of the program counter. The stack pointer then pops the value in location 0BDH (1AH) into the PC's low byte and the instruction at location 101AH is executed. The stack pointer now points to memory location 0BEH.

# RL — Rotate Left

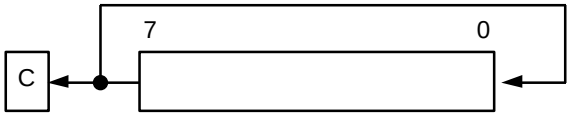
**RL** dst

**Operation:** C ¨ dst (7)

dst (0) ¨ dst (7)

dst (n + 1) ¨ dst (n), n = 0–6

The contents of the destination operand are rotated left one bit position. The initial value of bit 7 is moved to the bit zero (LSB) position and also replaces the carry flag.




**Flags:**
**C:** Set if the bit rotated from the most significant bit position (bit 7) was "1".
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result bit 7 is set; cleared otherwise.
**V:** Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
**D:** Unaffected.
**H:** Unaffected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst |
|---|---|---|---|---|
| opc dst | 2 | 4 | 90 | R |
| | | 4 | 91 | IR |

**Examples:** Given: Register 00H = 0AAH, register 01H = 02H and register 02H = 17H:

RL 00H → Register 00H = 55H, C = "1"
RL @01H → Register 01H = 02H, register 02H = 2EH, C = "0"

In the first example, if general register 00H contains the value 0AAH (10101010B), the statement "RL 00H" rotates the 0AAH value left one bit position, leaving the new value 55H (01010101B) and setting the carry and overflow flags.

# RLC — Rotate Left Through Carry

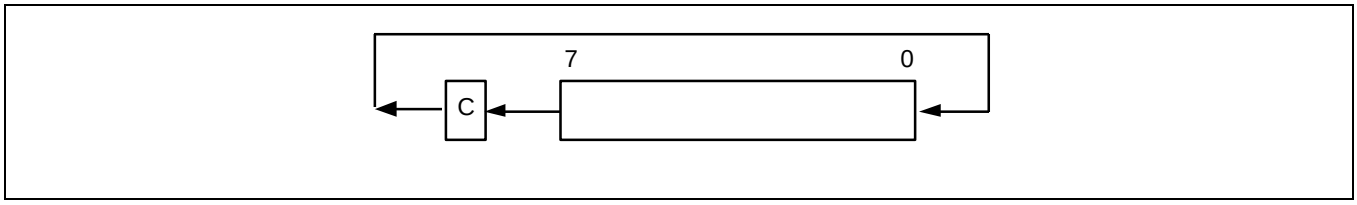
**RLC** dst

**Operation:** dst (0) ¨ C

C ¨ dst (7)

dst (n + 1) ¨ dst (n), n = 0–6

The contents of the destination operand with the carry flag are rotated left one bit position. The initial value of bit 7 replaces the carry flag (C); the initial value of the carry flag replaces bit zero.




**Flags:**
**C:** Set if the bit rotated from the most significant bit position (bit 7) was "1".
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result bit 7 is set; cleared otherwise.
**V:** Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
**D:** Unaffected.
**H:** Unaffected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst |
|---|---|---|---|---|
| opc dst | 2 | 4 | 10 | R |
| | | 4 | 11 | IR |

**Examples:** Given: Register 00H = 0AAH, register 01H = 02H, and register 02H = 17H, C = "0":

```
RLC   00H    →   Register 00H = 54H, C = "1"
RLC   @01H   →   Register 01H = 02H, register 02H = 2EH, C = "0"
```

In the first example, if general register 00H has the value 0AAH (10101010B), the statement "RLC 00H" rotates 0AAH one bit position to the left. The initial value of bit 7 sets the carry flag and the initial value of the C flag replaces bit zero of register 00H, leaving the value 55H (01010101B). The MSB of register 00H resets the carry flag to "1" and sets the overflow flag.

# RR — Rotate Right

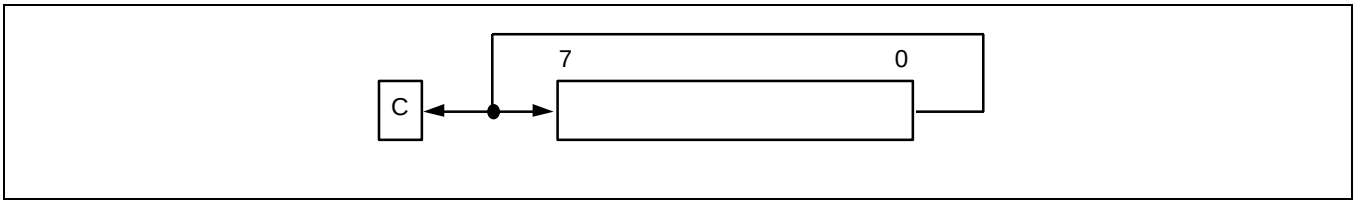
**RR** dst

**Operation:** C ¨ dst (0)

dst (7) ¨ dst (0)

dst (n) ¨ dst (n + 1), n = 0–6

The contents of the destination operand are rotated right one bit position. The initial value of bit zero (LSB) is moved to bit 7 (MSB) and also replaces the carry flag (C).




**Flags:**

**C:** Set if the bit rotated from the least significant bit position (bit zero) was "1".
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result bit 7 is set; cleared otherwise.
**V:** Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
**D:** Unaffected.
**H:** Unaffected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst |
|---|---|---|---|---|
| opc dst | 2 | 4 | E0 | R |
| | | 4 | E1 | IR |

**Examples:** Given: Register 00H = 31H, register 01H = 02H, and register 02H = 17H:

```
RR    00H    →    Register 00H = 98H, C = "1"
RR    @01H   →    Register 01H = 02H, register 02H = 8BH, C = "1"
```

In the first example, if general register 00H contains the value 31H (00110001B), the statement "RR 00H" rotates this value one bit position to the right. The initial value of bit zero is moved to bit 7, leaving the new value 98H (10011000B) in the destination register. The initial bit zero also resets the C flag to "1" and the sign flag and overflow flag are also set to "1".

# RRC — Rotate Right Through Carry

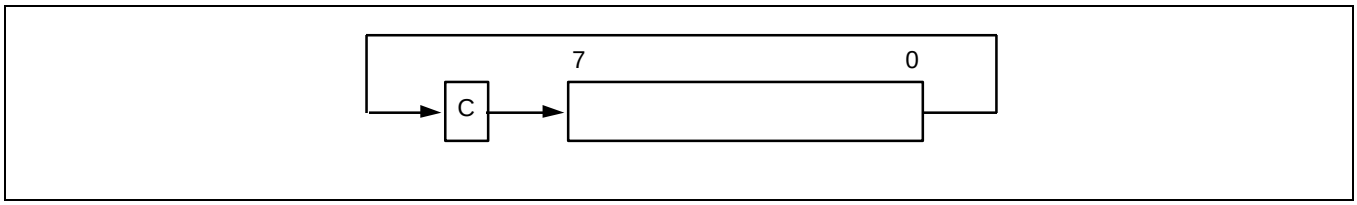
**RRC** dst

**Operation:** dst (7) ¨ C

C ¨ dst (0)

dst (n) ¨ dst (n + 1), n = 0–6

The contents of the destination operand and the carry flag are rotated right one bit position. The initial value of bit zero (LSB) replaces the carry flag; the initial value of the carry flag replaces bit 7 (MSB).




**Flags:**
**C:** Set if the bit rotated from the least significant bit position (bit zero) was "1".
**Z:** Set if the result is "0" cleared otherwise.
**S:** Set if the result bit 7 is set; cleared otherwise.
**V:** Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
**D:** Unaffected.
**H:** Unaffected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst |
|---|---|---|---|---|
| opc dst | 2 | 4 | C0 | R |
| | | 4 | C1 | IR |

**Examples:** Given: Register 00H = 55H, register 01H = 02H, register 02H = 17H, and C = "0":

RRC 00H → Register 00H = 2AH, C = "1"
RRC @01H → Register 01H = 02H, register 02H = 0BH, C = "1"

In the first example, if general register 00H contains the value 55H (01010101B), the statement "RRC 00H" rotates this value one bit position to the right. The initial value of bit zero ("1") replaces the carry flag and the initial value of the C flag ("1") replaces bit 7. This leaves the new value 2AH (00101010B) in destination register 00H. The sign flag and overflow flag are both cleared to "0".

# SBC — Subtract With Carry

**SBC** dst,src

**Operation:** dst ¨ dst – src – c

The source operand, along with the current value of the carry flag, is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. Subtraction is performed by adding the two's-complement of the source operand to the destination operand. In multiple precision arithmetic, this instruction permits the carry ("borrow") from the subtraction of the low-order operands to be subtracted from the subtraction of high-order operands.

**Flags:**

**C:** Set if a borrow occurred (src > dst); cleared otherwise.
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result is negative; cleared otherwise.
**V:** Set if arithmetic overflow occurred, that is, if the operands were of opposite sign and the sign f the result is the same as the sign of the source; cleared otherwise.
**D:** Always set to "1".
**H:** Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise, indicating a "borrow".

**Format:**

| | | | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | Addr Mode src |
|---|---|---|---|---|---|---|---|
| opc | dst \| src | | 2 | 4 | 32 | r | r |
| | | | | 6 | 33 | r | lr |
| opc | src | dst | 3 | 6 | 34 | R | R |
| | | | | 6 | 35 | R | IR |
| opc | dst | src | 3 | 6 | 36 | R | IM |

**Examples:** Given: R1 = 10H, R2 = 03H, C = "1", register 01H = 20H, register 02H = 03H, and register 03H = 0AH:

| | | | |
|---|---|---|---|
| SBC | R1,R2 | → | R1 = 0CH, R2 = 03H |
| SBC | R1,@R2 | → | R1 = 05H, R2 = 03H, register 03H = 0AH |
| SBC | 01H,02H | → | Register 01H = 1CH, register 02H = 03H |
| SBC | 01H,@02H | → | Register 01H = 15H,register 02H = 03H, register 03H = 0AH |
| SBC | 01H,#8AH | → | Register 01H = 95H; C, S, and V = "1" |

In the first example, if working register R1 contains the value 10H and register R2 the value 03H, the statement "SBC R1,R2" subtracts the source value (03H) and the C flag value ("1") from the destination (10H) and then stores the result (0CH) in register R1.

# SCF — Set Carry Flag

**SCF**

**Operation:** C ¨ 1

The carry flag (C) is set to logic one, regardless of its previous value.

**Flags:** **C:** Set to "1".

No other flags are affected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) |
|---|---|---|---|
| opc | 1 | 4 | DF |

**Example:** The statement

SCF

sets the carry flag to logic one.

# SRA — Shift Right Arithmetic

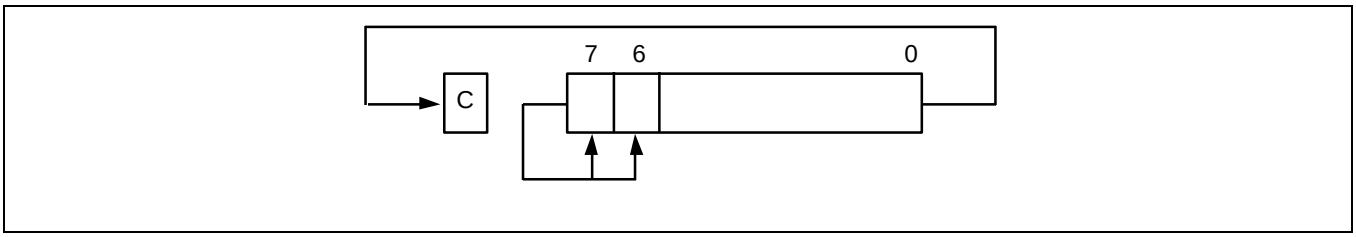
**SRA** dst

**Operation:** dst (7) ¨ dst (7)

C ¨ dst (0)

dst (n) ¨ dst (n + 1), n = 0–6

An arithmetic shift-right of one bit position is performed on the destination operand. Bit zero (the LSB) replaces the carry flag. The value of bit 7 (the sign bit) is unchanged and is shifted into bit position 6.




**Flags:**
**C:** Set if the bit shifted from the LSB position (bit zero) was "1".
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result is negative; cleared otherwise.
**V:** Always cleared to "0".
**D:** Unaffected.
**H:** Unaffected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst |
|---|---|---|---|---|
| opc dst | 2 | 4 | D0 | R |
| | | 4 | D1 | IR |

**Examples:** Given: Register 00H = 9AH, register 02H = 03H, register 03H = 0BCH, and C = "1":

```
SRA     00H     →     Register 00H = 0CD, C = "0"
SRA     @02H    →     Register 02H = 03H, register 03H = 0DEH, C = "0"
```

In the first example, if general register 00H contains the value 9AH (10011010B), the statement "SRA 00H" shifts the bit values in register 00H right one bit position. Bit zero ("0") clears the C flag and bit 7 ("1") is then shifted into the bit 6 position (bit 7 remains unchanged). This leaves the value 0CDH (11001101B) in destination register 00H.

# STOP — Stop Operation

**STOP**

**Operation:**

The STOP instruction stops both the CPU clock and system clock and causes the microcontroller to enter Stop mode. During Stop mode, the contents of on-chip CPU registers, peripheral registers, and I/O port control and data registers are retained. Stop mode can be released by an external reset operation or External interrupt input. For the reset operation, the RESET pin must be held to Low level until the required oscillation stabilization interval has elapsed.

**Flags:** No flags are affected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | src |
|---|---|---|---|---|---|
| opc | 1 | 4 | 7F | – | – |

**Example:** The statement

STOP

halts all microcontroller operations.

# SUB — Subtract

**SUB** dst,src

**Operation:** dst ¨ dst – src

The source operand is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. Subtraction is performed by adding the two's complement of the source operand to the destination operand.

**Flags:**
**C:** Set if a "borrow" occurred; cleared otherwise.
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result is negative; cleared otherwise.
**V:** Set if arithmetic overflow occurred, that is, if the operands were of opposite signs and the sign of the result is of the same as the sign of the source operand; cleared otherwise.
**D:** Always set to "1".
**H:** Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise indicating a "borrow".

**Format:**

| | | | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | Addr Mode src |
|---|---|---|---|---|---|---|---|
| opc | dst \| src | | 2 | 4 | 22 | r | r |
| | | | | 6 | 23 | r | lr |
| opc | src | dst | 3 | 6 | 24 | R | R |
| | | | | 6 | 25 | R | IR |
| opc | dst | src | 3 | 6 | 26 | R | IM |

**Examples:** Given: R1 = 12H, R2 = 03H, register 01H = 21H, register 02H = 03H, register 03H = 0AH:

| | | | |
|---|---|---|---|
| SUB | R1,R2 | → | R1 = 0FH, R2 = 03H |
| SUB | R1,@R2 | → | R1 = 08H, R2 = 03H |
| SUB | 01H,02H | → | Register 01H = 1EH, register 02H = 03H |
| SUB | 01H,@02H | → | Register 01H = 17H, register 02H = 03H |
| SUB | 01H,#90H | → | Register 01H = 91H; C, S, and V = "1" |
| SUB | 01H,#65H | → | Register 01H = 0BCH; C and S = "1", V = "0" |

In the first example, if working register R1 contains the value 12H and if register R2 contains the value 03H, the statement "SUB R1,R2" subtracts the source value (03H) from the destination value (12H) and stores the result (0FH) in destination register R1.

# TCM — Test Complement Under Mask

**TCM** dst,src

**Operation:** (NOT dst) AND src

This instruction tests selected bits in the destination operand for a logic one value. The bits to be tested are specified by setting a "1" bit in the corresponding position of the source operand (mask). The TCM statement complements the destination operand, which is then ANDed with the source mask. The zero (Z) flag can then be checked to determine the result. The destination and source operands are unaffected.

**Flags:**
**C:** Unaffected.
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result bit 7 is set; cleared otherwise.
**V:** Always cleared to "0".
**D:** Unaffected.
**H:** Unaffected.

**Format:**

| | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | src |
|---|---|---|---|---|---|
| opc \| dst \| src | 2 | 4 | 62 | r | r |
| | | 6 | 63 | r | lr |
| opc \| src \| dst | 3 | 6 | 64 | R | R |
| | | 6 | 65 | R | IR |
| opc \| dst \| src | 3 | 6 | 66 | R | IM |

**Examples:** Given: R0 = 0C7H, R1 = 02H, R2 = 12H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

```
TCM   R0,R1      →   R0 = 0C7H, R1 = 02H, Z = "1"
TCM   R0,@R1     →   R0 = 0C7H, R1 = 02H, register 02H = 23H, Z = "0"
TCM   00H,01H    →   Register 00H = 2BH, register 01H = 02H, Z = "1"
TCM   00H,@01H   →   Register 00H = 2BH, register 01H = 02H,
                     register 02H = 23H, Z = "1"
TCM   00H,#34    →   Register 00H = 2BH, Z = "0"
```

In the first example, if working register R0 contains the value 0C7H (11000111B) and register R1 the value 02H (00000010B), the statement "TCM R0,R1" tests bit one in the destination register for a "1" value. Because the mask value corresponds to the test bit, the Z flag is set to logic one and can be tested to determine the result of the TCM operation.

# TM — Test Under Mask

**TM** dst,src

**Operation:** dst AND src

This instruction tests selected bits in the destination operand for a logic zero value. The bits to be tested are specified by setting a "1" bit in the corresponding position of the source operand (mask), which is ANDed with the destination operand. The zero (Z) flag can then be checked to determine the result. The destination and source operands are unaffected.

**Flags:**
**C:** Unaffected.
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result bit 7 is set; cleared otherwise.
**V:** Always reset to "0".
**D:** Unaffected.
**H:** Unaffected.

**Format:**

| | | | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | Addr Mode src |
|---|---|---|---|---|---|---|---|
| opc | dst \| src | | 2 | 4 | 72 | r | r |
| | | | | 6 | 73 | r | lr |
| opc | src | dst | 3 | 6 | 74 | R | R |
| | | | | 6 | 75 | R | IR |
| opc | dst | src | 3 | 6 | 76 | R | IM |

**Examples:** Given: R0 = 0C7H, R1 = 02H, R2 = 18H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

| | | | |
|---|---|---|---|
| TM | R0,R1 | → | R0 = 0C7H, R1 = 02H, Z = "0" |
| TM | R0,@R1 | → | R0 = 0C7H, R1 = 02H, register 02H = 23H, Z = "0" |
| TM | 00H,01H | → | Register 00H = 2BH, register 01H = 02H, Z = "0" |
| TM | 00H,@01H | → | Register 00H = 2BH, register 01H = 02H, register 02H = 23H, Z = "0" |
| TM | 00H,#54H | → | Register 00H = 2BH, Z = "1" |

In the first example, if working register R0 contains the value 0C7H (11000111B) and register R1 the value 02H (00000010B), the statement "TM R0,R1" tests bit one in the destination register for a "0" value. Because the mask value does not match the test bit, the Z flag is cleared to logic zero and can be tested to determine the result of the TM operation.

# XOR — Logical Exclusive OR

**XOR** dst,src

**Operation:** dst ¨ dst XOR src

The source operand is logically exclusive-ORed with the destination operand and the result is stored in the destination. The exclusive-OR operation results in a "1" bit being stored whenever the corresponding bits in the operands are different; otherwise, a "0" bit is stored.

**Flags:**
**C:** Unaffected.
**Z:** Set if the result is "0"; cleared otherwise.
**S:** Set if the result bit 7 is set; cleared otherwise.
**V:** Always reset to "0".
**D:** Unaffected.
**H:** Unaffected.

**Format:**

| | | | Bytes | Cycles | Opcode (Hex) | Addr Mode dst | Addr Mode src |
|---|---|---|---|---|---|---|---|
| opc | dst \| src | | 2 | 4 | B2 | r | r |
| | | | | 6 | B3 | r | lr |
| opc | src | dst | 3 | 6 | B4 | R | R |
| | | | | 6 | B5 | R | IR |
| opc | dst | src | 3 | 6 | B6 | R | IM |

**Examples:** Given: R0 = 0C7H, R1 = 02H, R2 = 18H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

| | | | |
|---|---|---|---|
| XOR | R0,R1 | → | R0 = 0C5H, R1 = 02H |
| XOR | R0,@R1 | → | R0 = 0E4H, R1 = 02H, register 02H = 23H |
| XOR | 00H,01H | → | Register 00H = 29H, register 01H = 02H |
| XOR | 00H,@01H | → | Register 00H = 08H, register 01H = 02H, register 02H = 23H |
| XOR | 00H,#54H | → | Register 00H = 7FH |

In the first example, if working register R0 contains the value 0C7H and if register R1 contains the value 02H, the statement "XOR R0,R1" logically exclusive-ORs the R1 value with the R0 value and stores the result (0C5H) in the destination register R0.

# 7 CLOCK CIRCUIT

## OVERVIEW

The S3C9654/C9658/P9658 has three oscillation circuit options, a crystal/ceramic oscillation and a RC oscillation and an external clock source. The crystal or ceramic oscillation source provides a maximum 6 MHz clock. The $X_{IN}$ and $X_{OUT}$ pins connect the oscillation source to the on-chip clock circuit. External clock and RC oscillation and crystal/ceramic oscillator circuits are shown in Figures 7-1, 7-2, and 7-3.

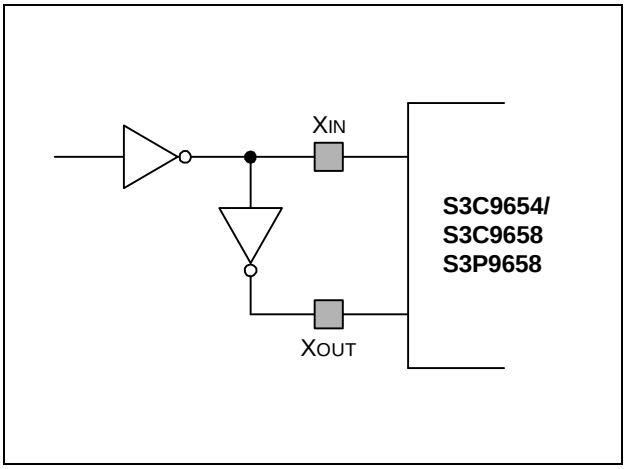



**Figure 7-1. External Oscillator**

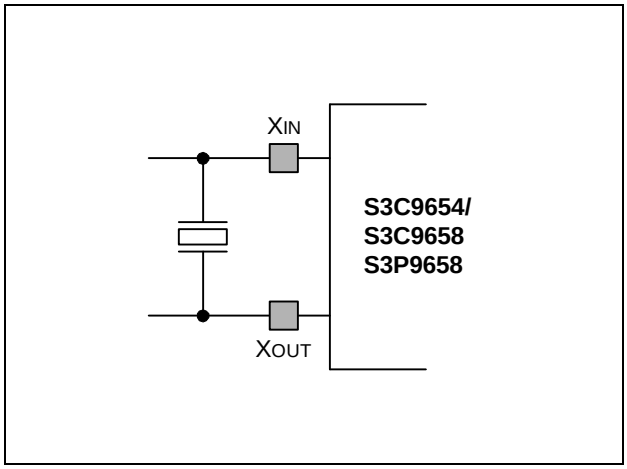



**Figure 7-2. Main Oscillator Circuit (Crystal/Ceramic Oscillator)**

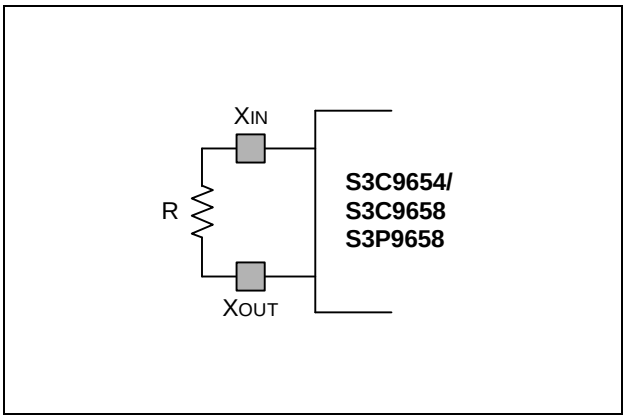



**Figure 7-3. RC Oscillator**

## MAIN OSCILLATOR LOGIC

To increase processing speed and to reduce clock noise, non-divided logic is implemented for the main oscillator circuit. For this reason, very high resolution waveforms (square signal edges) must be generated in order for the CPU to efficiently process logic operations.

## CLOCK STATUS DURING POWER-DOWN MODES

The two power-down modes, Stop mode and Idle mode, affect clock oscillation as follows:

— In Stop mode, the main oscillator "freezes," halting the CPU and peripherals. The contents of the register file and current system register values are retained. RESET operation releases the Stop mode, and starts the oscillator.

— In Idle mode, the internal clock signal is gated off to the CPU, but not to interrupt control and the timer. The current CPU status is preserved, including stack pointer, program counter, and flags. Data in the register file is retained. Idle mode is released by a RESET or by an interrupt (external or internally-generated).

## SYSTEM CLOCK CONTROL REGISTER (CLKCON)

The system clock control register, CLKCON, is located in location D4H. It is read/write addressable and has the following functions:

— Oscillator IRQ wake-up function enable/disable (CLKCON.7)

— Oscillator frequency divide-by value: non-divided, 2, 8, or 16 (CLKCON.4 and CLKCON.3)

The CLKCON register controls whether or not an external interrupt can be used to trigger a Stop mode release (This is called the "IRQ wake-up" function). The IRQ wake-up enable bit is CLKCON.7.

After a RESET, the external interrupt oscillator wake-up function is enabled, the main oscillator is activated, and the $f_{OSC}/16$ (the slowest clock speed) is selected as the CPU clock. If necessary, you can then increase the CPU clock speed to $f_{OSC}$, $f_{OSC}/2$ or $f_{OSC}/8$.

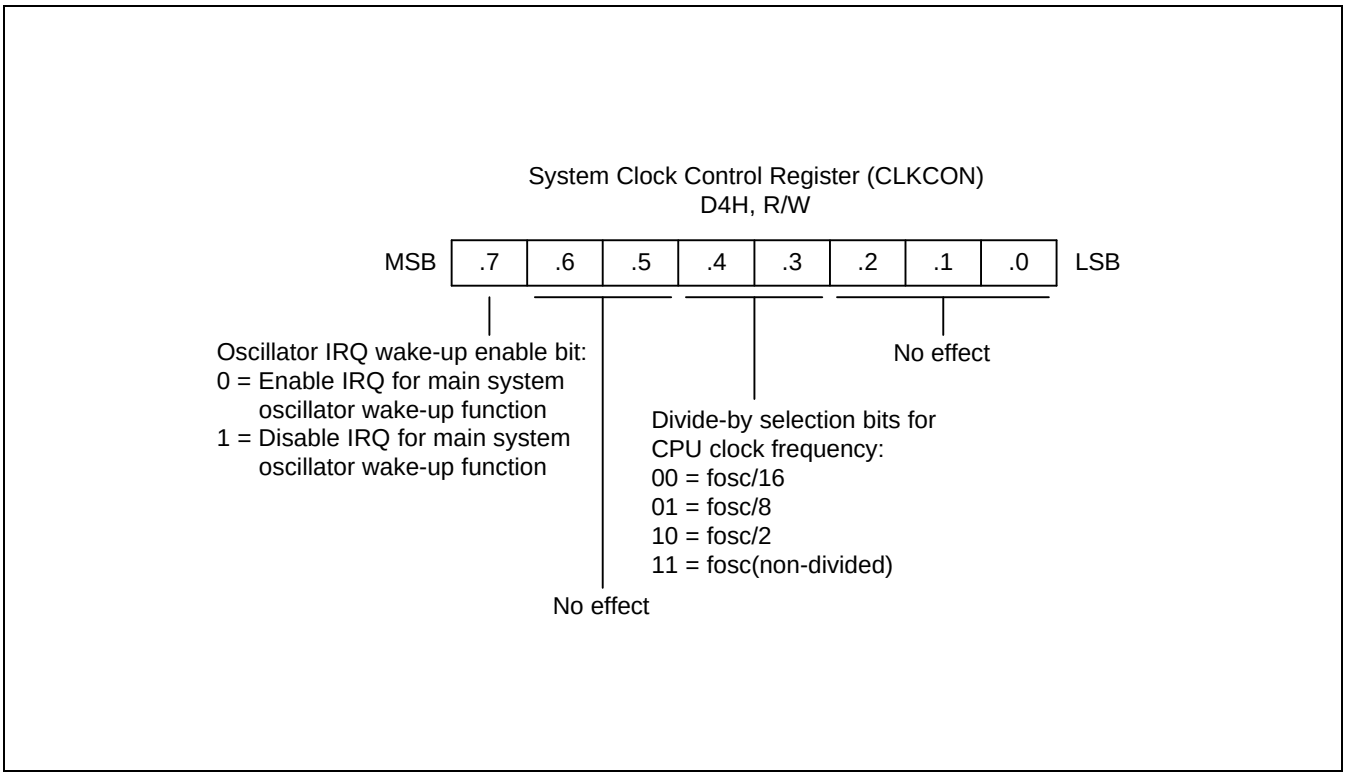



**Figure 7-4. System Clock Control Register (CLKCON)**

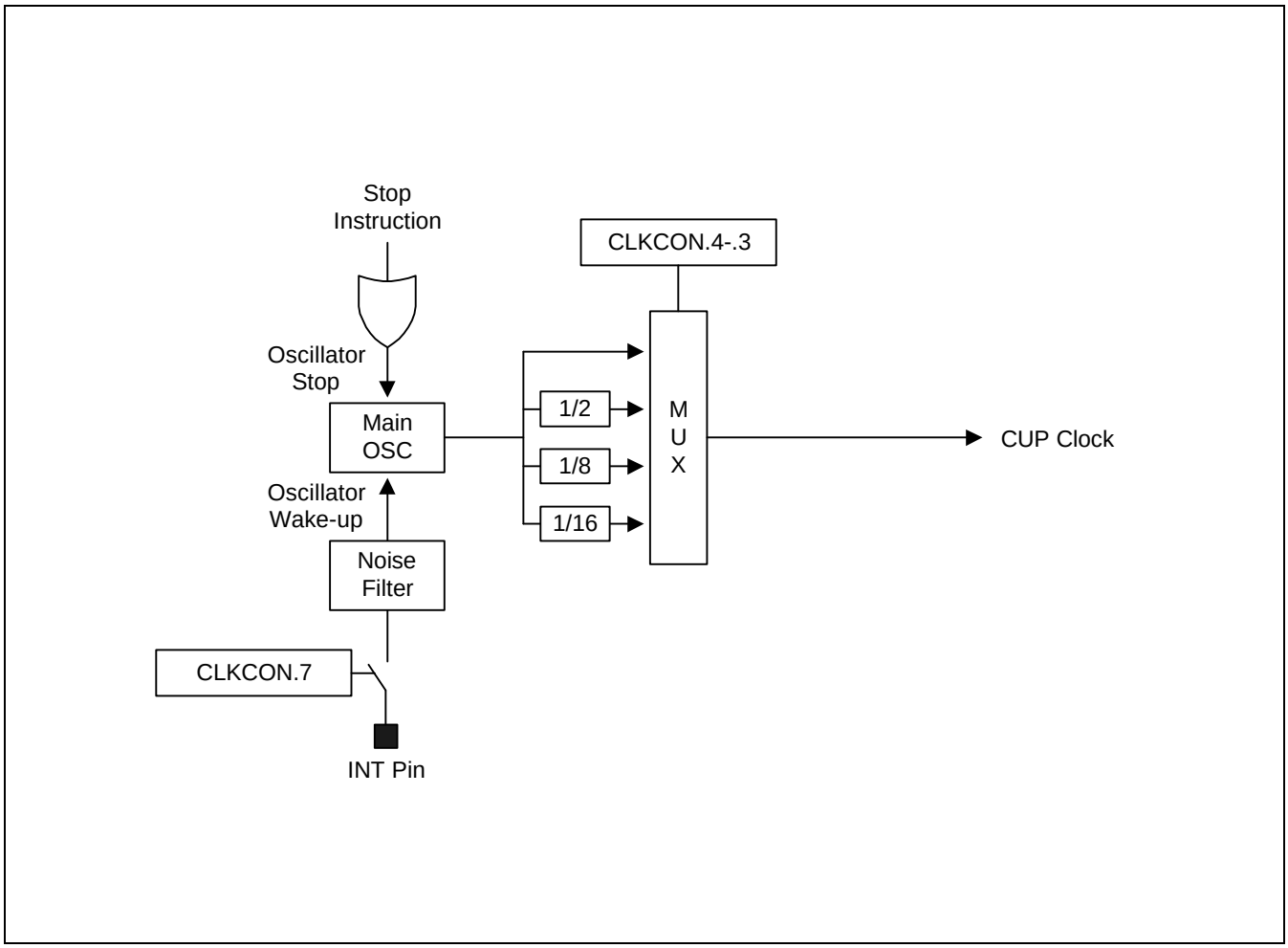
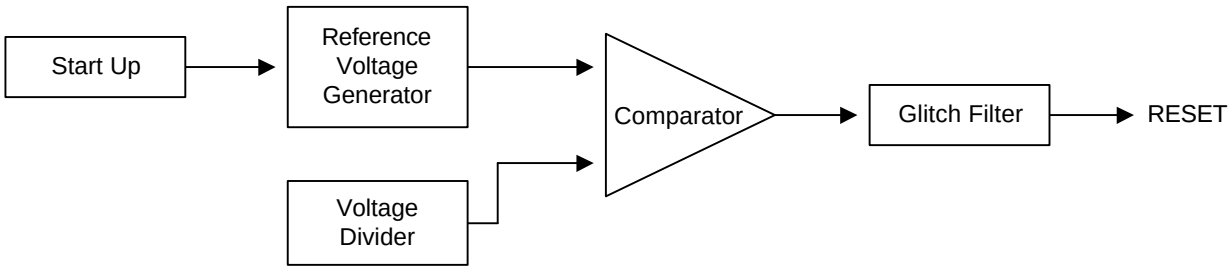



**Figure 7-5. System Clock Circuit Diagram**

# 8 RESET and POWER-DOWN

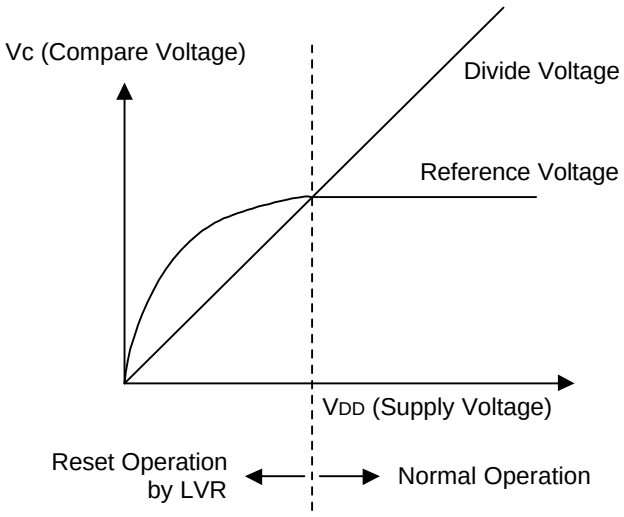
## SYSTEM RESET

### OVERVIEW




**NOTES:**
1. Start Up Circuit: Start up reference voltage generator circuit when device powered.
2. Reference Voltage Generator: Supply voltage independent reference voltage generator. (Supply voltage must great then 2.5 V)
3. Voltage Divider: Divide supply voltage by "N" (N: integer, 2).
4. Comparator: Compare reference voltage and divided voltage.
5. Glitch Filter: Remove glitch and noise signal.

**Figure 8-1. LVR (LVD) Architecture**




**NOTES:**

1. LVR Operation Voltage Range: 2.3 V-6.0 V
2. LVR Detection Voltage Range: 3.4 V ±0.4 V
3. LVR Current Consumption:
   Less then 10 uA (normally 5 uA)
4. LVR Powered Reset Release Time:
   more then 500 usec (LVR only, typical)
5. LVR Simulation Conditions (Hspice Simulation)
   Temp: -40 - 80 °C
   Process Veriation: Worst to best conditions
   Test Voltage: 0.0 V-7.0 V
   Powered Slew Rate: 5 V/1 usec- 5 V/100 msec

**Figure 8-2. LVR Characteristics**

The following sequence of events occur during a RESET operation:

— All interrupts are disabled.
— The watchdog function (basic timer) is enabled.
— Ports 0 and 1 are set to Schmitt trigger input mode and all pull-up resistors are disabled.
— Peripheral control and data registers are disabled and RESET to their initial values.
— The program counter is loaded with the ROM RESET address, 0100H.
— When the programmed oscillation stabilization time interval has elapsed, the address stored in ROM location 0100H (and 0101H) is fetched and executed.

**NOTE**

To program the duration of the oscillation stabilization interval, you must make the appropriate settings to the basic timer control register, BTCON, before entering Stop mode. Also, if you do not want to use the basic timer watchdog function (which causes a system RESET if a basic timer counter overflow occurs), you can disable it by writing '1010B' to the upper nibble of BTCON.

# POWER-DOWN MODES

## STOP MODE

Stop mode is invoked by the instruction STOP (opcode 7FH). In Stop mode, the operation of the CPU and all peripherals is halted. That is, the on-chip main oscillator stops and the supply current is reduced to less than 120 µA. All system functions are halted when the clock "freezes," but data stored in the internal register file is retained. Stop mode can be released in one of two ways: by a RESET signal or by an external interrupt.

### Using RESET to Release Stop Mode

Stop mode is released when the RESET signal is released and returns to High level. All system and peripheral control registers are then RESET to their default values and the contents of all data registers are retained. RESET operation automatically selects a slow clock (1/16) because CLKCON.3 and CLKCON.4 are cleared to '00B'. After the oscillation stabilization interval has elapsed, the CPU executes the system initialization routine by fetching the 16-bit address stored in ROM locations 0100H and 0101H.

### Using an External Interrupt to Release Stop Mode

Only external interrupts with an RC-delay noise filter circuit can be used to release Stop mode (Clock-related external interrupts cannot be used). External interrupts in the KS86C6504/P6508 interrupt structure does not meet this criteria.

Note that when Stop mode is released by an external interrupt, the current values in system and peripheral control registers are not changed. When you use an interrupt to release Stop mode, the CLKCON.3 and CLKCON.4 register values remain unchanged, and the currently selected clock value is used. If you use an external interrupt for Stop mode release, you can also program the duration of the oscillation stabilization interval. To do this, you must make the appropriate control and clock settings *before* entering Stop mode.

The external interrupt is serviced when the Stop mode release occurs. Following the IRET from the service routine, the instruction immediately following the one that initiated Stop mode is executed.

**NOTE**

Do not use the STOP mode when external clock source is being used as the oscillation circuit option.

## IDLE MODE

Idle mode is invoked by the instruction IDLE (opcode 6FH). In Idle mode, CPU operations are halted while select peripherals remain active. During Idle mode, the internal clock signal is gated off to the CPU, but not to interrupt logic and timer/counters. Port pins retain the mode (input or output) they had at the time Idle mode was entered.

There are two ways to release Idle mode:

1. Execute RESET. All system and peripheral control registers are RESET to their default values and the contents of all data registers are retained. The RESET automatically selects a slow clock (1/16) because CLKCON.3 and CLKCON.4 are cleared to '00B'. If interrupts are masked, RESET is the only way to release Idle mode.
2. Activate any enabled interrupt, causing Idle mode to be released. When you use an interrupt to release Idle mode, the CLKCON.3 and CLKCON.4 register values remain unchanged, and the currently selected clock value is used. The interrupt is then serviced. Following the IRET from the service routine, the instruction immediately following the one that initiated Idle mode is executed.

**NOTE**

Only external interrupts that are not clock-related can be used to release Stop mode. To release Idle mode, however, any type of interrupt (that is, internal or external) can be used.

## HARDWARE RESET VALUES

Tables 8-1 through 8-3 list the values for CPU and system registers, peripheral control registers, and peripheral data registers following a RESET operation in normal operating mode. The following notation is used in these tables to represent specific RESET values:

— A "1" or a "0" shows the RESET bit value as logic one or logic zero, respectively.

— An 'x' means that the bit value is undefined following RESET.

— A dash ('–') means that the bit is either not used or not mapped.

**Table 8-1. Register Values after RESET**

| Register Name | Mnemonic | Address | Bit Values After RESET | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| General purpose register file & stack area | – | 00–7FH | x | x | x | x | x | x | x | x |
| Working register area | – | C0H–CFH | x | x | x | x | x | x | x | x |
| Timer 0 counter register | T0CNT | D0H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Timer 0 data register | T0DATA | D1H | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Timer 0 control register | T0CON | D2H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Location D3H is not mapped. | | | | | | | | | | |
| Clock control register | CLKCON | D4H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| System FLAG register | FLAGS | D5H | 0 | 0 | 0 | 0 | – | – | – | – |
| Locations D6H–D8H are not mapped. | | | | | | | | | | |
| Stack pointer | SP | D9H | – | – | – | – | – | – | – | – |
| Locations DAH–DBH are not mapped. | | | | | | | | | | |
| Basic timer control register | BTCON | DCH | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Basic timer counter register | BTCNT | DDH | – | – | – | – | – | – | – | – |
| Location DEH is not mapped. | | | | | | | | | | |
| System mode register | SYM | DFH | – | – | – | – | 0 | 0 | 0 | 0 |

**NOTE:** The timer 0 counter, T0CNT, the basic timer counter, BTCNT, and comparison result, CDATA, are read-only. All other registers are read/write addressable.

**Table 8-1. Register Values after RESET (Continued)**

| Bank 0 Register Name | Mnemonic | Address | Bit Values After RESET | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Port 0 data register | P0 | E0H | x | x | 0 | 0 | 0 | 0 | 0 | 0 |
| Port 1 data register | P1 | E1H | x | x | 0 | 0 | 0 | 0 | 0 | 0 |
| Port 2 data register | P2 | E2H | x | x | x | x | x | x | 0 | 0 |
| Port 1 pull-down control | PDCON | E3H | x | x | x | x | 0 | 0 | 0 | 0 |
| Comparator control mode register | CCON | E4H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Comparison result register | CDATA | E5H | x | x | 0 | 0 | 0 | 0 | 0 | 0 |
| Port 0 low nibble control register | P0CONL | E6H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Port 0 high bit control register | P0CONH | E7H | x | x | x | x | 0 | 0 | 0 | 0 |
| Port 1 high bit control register | P1CONH | E8H | x | x | x | x | 0 | 0 | 0 | 0 |
| Port 1 low nibble control register | P1CONL | E9H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Port 0 interrupt control register | P0INT | EAH | x | x | 0 | 0 | 0 | 0 | 0 | 0 |
| Port 0 interrupt pending register | P0PND | EBH | x | x | 0 | 0 | 0 | 0 | 0 | 0 |
| Port 1 interrupt control register | P1INT | ECH | x | x | 0 | 0 | 0 | 0 | 0 | 0 |
| Port 1 interrupt pending register | P1PND | EDH | x | x | 0 | 0 | 0 | 0 | 0 | 0 |
| Port 2 control/interrupt control and pending register | P2CONINT | EEH | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Sub oscillator control register | SUBCON | EFH | x | 0 | x | x | 0 | 0 | 0 | 0 |
| USB function address register | FADDR | F0H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Control endpoint status register | EP0CSR | F1H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Interrupt endpoint status register | EP1CSR | F2H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Control endpoint byte count register | EP0BCNT | F3H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Control endpoint FIFO register | EP0FIFO | F4H | x | x | x | x | x | x | x | x |
| Interrupt endpoint FIFO register | EP1FIFO | F5H | x | x | x | x | x | x | x | x |
| USB interrupt pending register | USBPND | F6H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| USB interrupt enable register | USBINT | F7H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| USB power management register | PWRMGR | F8H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Locations F9H–FAH are not mapped. | | | | | | | | | | |
| USB mode select register | USBSEL | FBH | x | x | x | x | x | x | x | 0 |
| Locations FCH is not mapped. | | | | | | | | | | |
| Sink current control register | SNKCON | FDH | x | x | x | x | x | x | 0 | 0 |
| USB signal control | XCON | FEH | x | x | 0 | 0 | 0 | 0 | 0 | 0 |
| USB RESET register | USBRST | FFH | x | x | x | x | x | x | x | 0 |

# NOTES

# 9 I/O PORTS

## OVERVIEW

The S3C9654/C9658/P9658 has two I/O ports (Port 0, Port 1, Port 2 at PS/2 Mode only), 14 pins total. You access these ports directly by writing or reading port data register addresses.

For mouse applications, ports 1.0–1.5 are usually configured as mouse sensing input. Port 0 is used for button data input.

**Table 9-1. S3C9654/C9658/P9658 Port Configuration Overview**

| Port | Function Description | Programmability |
|---|---|---|
| P0.0 | Bit-programmable I/O port for Schmitt trigger input or n-ch open drain output (50 mA). Pull-up resistor is assignable to input pin by software and is automatically disabled for output pin.<br>Port 0 can be individualy configured as external interrupt input. | Bit |
| P0.1 – P0.5 | Bit-programmable I/O port for Schmitt trigger input or push-pull output. Pull-up resistors are individually assignable to input pins by software and are automatically disabled for output pins.<br>Port 0 can be individualy configured as external interrupt inputs. | Bit |
| P1.0 – P1.5 | Bit-programmable I/O port for Schmitt trigger input or push-pull output. Pull-up resistors are individually assignable to input pins by software.<br>Port 1 can be configured as comparator input or external interrupt inputs. Pull-down resistors are individually assignable. (in comparator input). | Bit |
| P2.0/D-<br>–<br>P2.1/D+ | Bit-programmable I/O port for Schmitt trigger input or n-ch open drain output. Pull-up resistors are individually assignable to input pins by software and are automatically disabled for output pins.<br>Port 2 can be individually configured as external interrupt pins.<br>Also it can be configured as an USB ports. | Bit |

## PORT DATA REGISTERS

Table 9-2 gives you an overview of the port data register names, locations, and addressing characteristics. Data registers for ports 0 and 1 have the structure shown in Figure 9-1.

**Table 9-2. Port Data Register Summary**

| Register Name | Mnemonic | Hex | R/W |
|---|---|---|---|
| Port 0 data register | P0 | E0H | R/W |
| Port 1 data register | P1 | E1H | R/W |
| Port 2 data register | P2 | E2H | R/W |

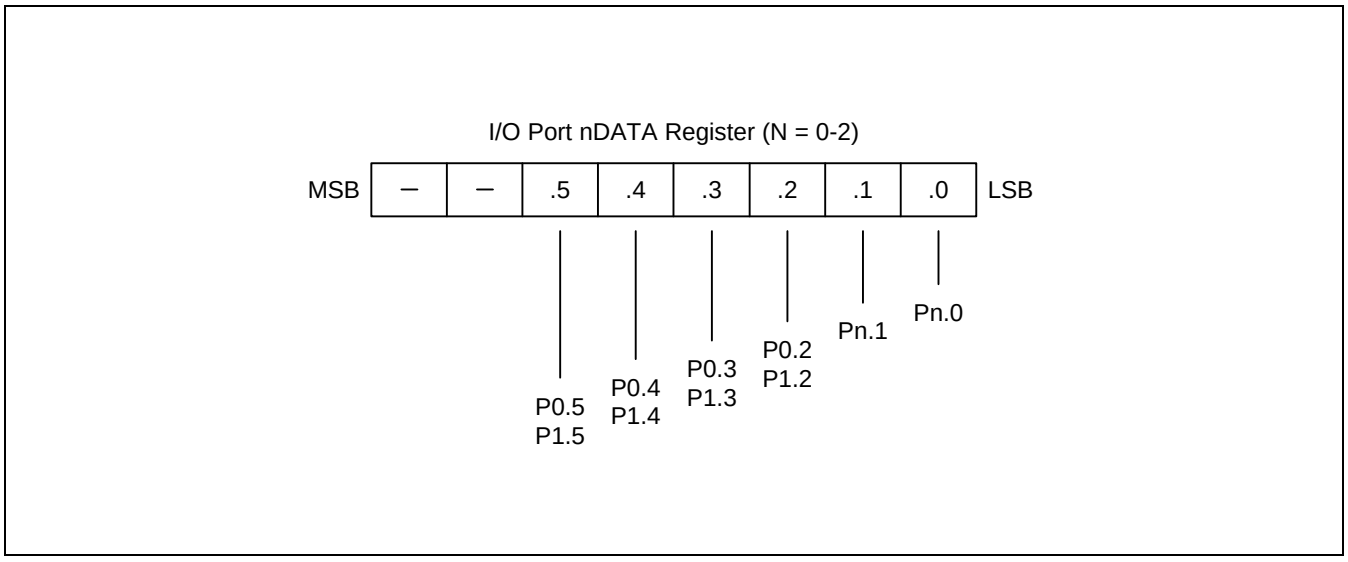
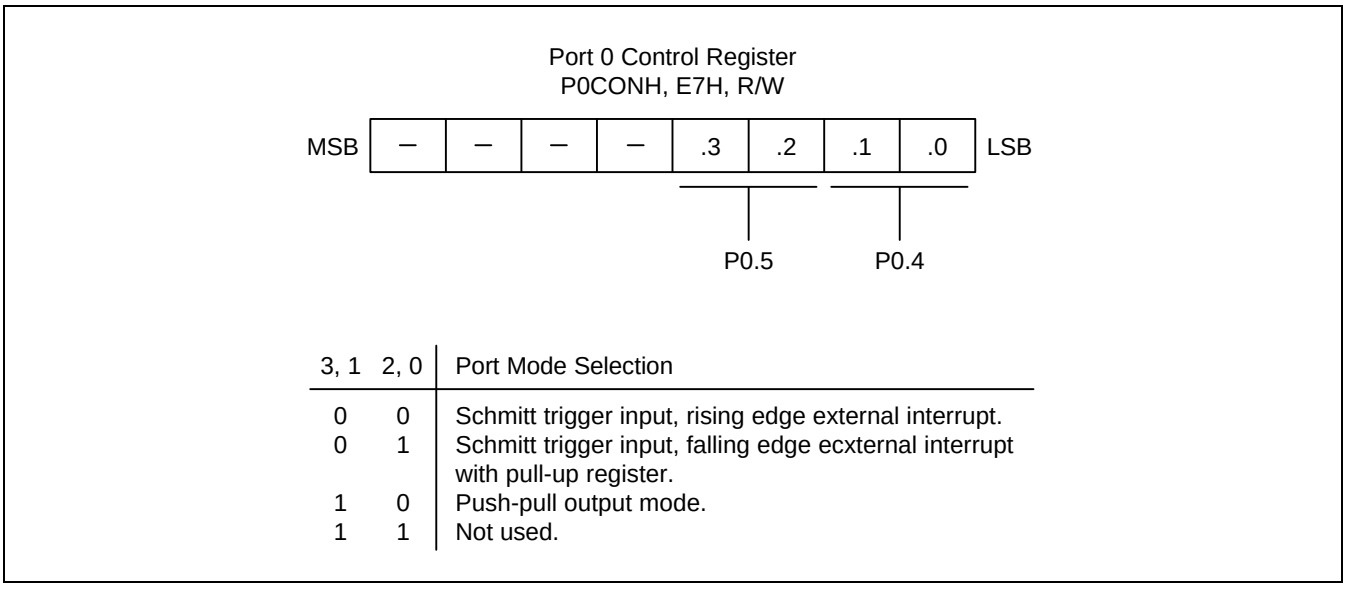



**Figure 9-1. Port Data Register Format**

## PORT 0, PORT 1 AND PORT 2

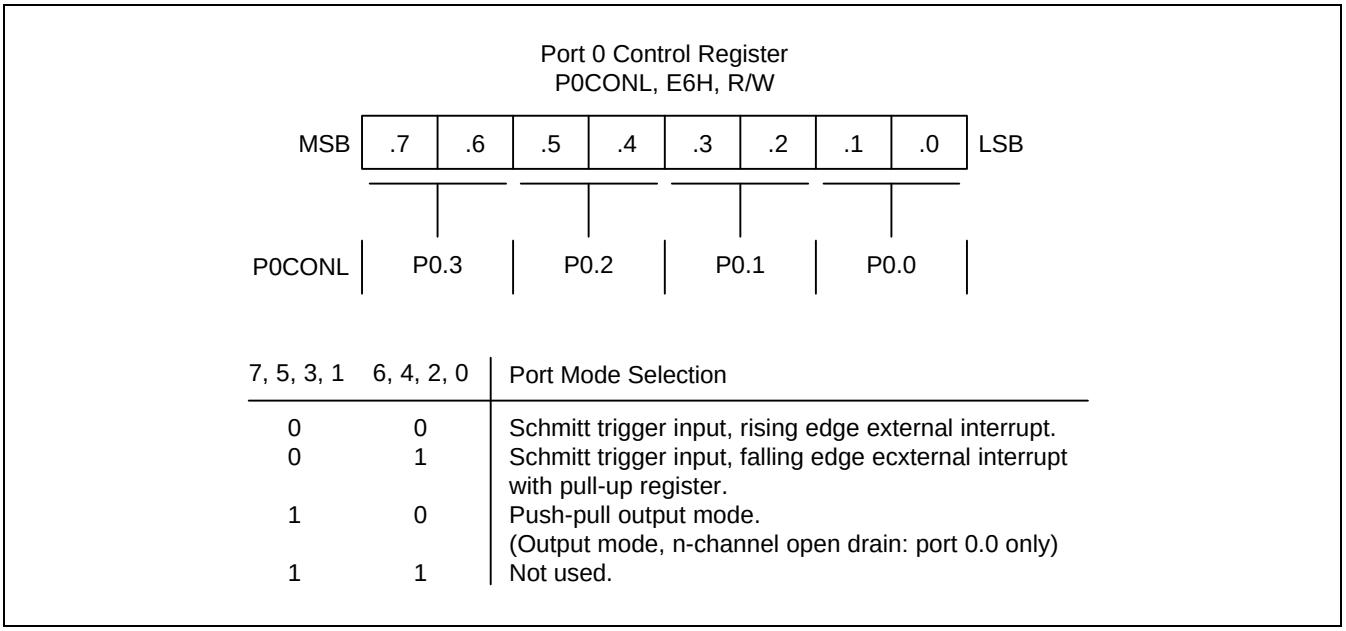
Ports 0, 1 and 2 are bit-programmable, general-purpose, I/O ports. You can select Schmitt trigger input mode with rising edge external interrupt or push-pull output mode. Port1.0 to Port1.5 can be configured as comparator input.

You access ports 0, 1 and 2 directly by writing or reading the corresponding port data registers — P0 (E0H), P1 (E1H) and P2 (E2H). RESET clears the port control registers P0CONH, P0CONL, P1CONH, P1CONL and P2CONINT, to '00H', configuring all port 0, port 1, port 2 pins as schmitt trigger inputs.




| 3, 1 | 2, 0 | Port Mode Selection |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge ecxternal interrupt with pull-up register. |
| 1 | 0 | Push-pull output mode. |
| 1 | 1 | Not used. |

**Figure 9-2. Port 0 Control Registers (P0CONH)**




| 7, 5, 3, 1 | 6, 4, 2, 0 | Port Mode Selection |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge ecxternal interrupt with pull-up register. |
| 1 | 0 | Push-pull output mode. (Output mode, n-channel open drain: port 0.0 only) |
| 1 | 1 | Not used. |

**Figure 9-3. Port 0 Control Registers (P0CONL)**

Port 1 Control Registers
P1CONH, E8H, R/W, P1CONL, E9H, R/W

| MSB | .7 | .6 | .5 | .4 | .3 | .2 | .1 | .0 | LSB |
|---|---|---|---|---|---|---|---|---|---|
| P1CONH | – | | – | | P1.5 | | P1.4 | | |
| P1CONL | P1.3 | | P1.2 | | P1.1 | | P1.0 | | |

| 7, 5, 3, 1 | 6, 4, 2, 0 | Port Mode Selection |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge external interrupt with pull-up register. |
| 1 | 0 | Push-pull output mode. |
| 1 | 1 | Comparator input. |

**Figure 9-4. Control Registers (P1CONH, P1CONL)**

### ☞ PROGRAMMING TIP — Configuring S3C9654/C9658/P9658 Port Pins to Specification

This example shows how to configure ports 0–1 to specification. The programming parameters are as follows:

**Examples:** 1. Set port 0 push-pull output mode

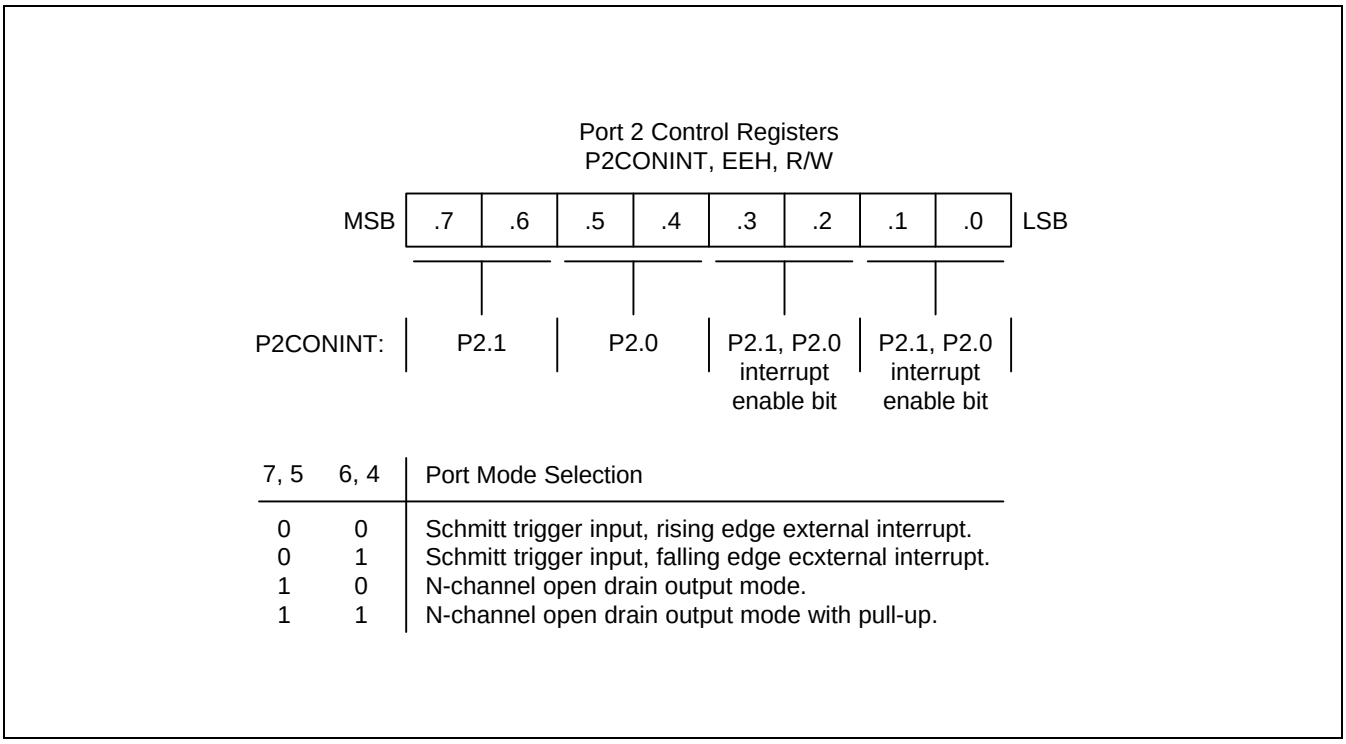
```
LD  P0CONL,#0AAH        ;  P0.1–P0.3 ← Push-pull output (P0.0 ← Open-drain output)
```

2. Set port 1.4–port 1.5 schmitt trigger input mode

```
LD  P1CONH,#00H         ;  P1.4–P1.5 ← Schmitt trigger input
```

3. Set port 1.0–port 1.3 comparator input mode

```
LD  P1CONL,#0FFH        ;  P1.0–P1.3 ← Comparator input
```




| 7, 5 | 6, 4 | Port Mode Selection |
|---|---|---|
| 0 | 0 | Schmitt trigger input, rising edge external interrupt. |
| 0 | 1 | Schmitt trigger input, falling edge ecxternal interrupt. |
| 1 | 0 | N-channel open drain output mode. |
| 1 | 1 | N-channel open drain output mode with pull-up. |

**Figure 9-5. Port Control Registers (P2CONINT)**

# NOTES

# 10 BASIC TIMER and TIMER 0

## MODULE OVERVIEW

The S3C9654/C9658/P9658 has two default timers: an 8-bit *basic timer* and one 8-bit general-purpose timer/counter. The 8-bit timer/counter is called timer 0.

### Basic Timer (BT)

You can use the basic timer (BT) in two different ways:

— As a watchdog timer to provide an automatic reset mechanism in the event of a system malfunction.
— To signal the end of the required oscillation stabilization interval after a reset or a Stop mode release.

The functional components of the basic timer block are:

— Clock frequency divider ($f_{OSC}$ divided by 4096, 1024, or 128) with multiplexer
— 8-bit basic timer counter, BTCNT (DDH, read-only)
— Basic timer control register, BTCON (DCH, read/write)

### Timer 0

Timer 0 has two operating modes, one of which you select by the appropriate T0CON setting:

— Interval timer mode
— Overflow mode

Timer 0 has the following functional components:

— Clock frequency divider ($f_{OSC}$ divided by 4096, 256, or 8) with multiplexer
— 8-bit counter (T0CNT), 8-bit comparator, and 8-bit reference data register (T0DATA)
— Timer 0 overflow interrupt (T0OVF) and match interrupt (T0INT) generation
— Timer 0 control register, T0CON

## BASIC TIMER CONTROL REGISTER (BTCON)

The basic timer control register, BTCON, is used to select the input clock frequency, to clear the basic timer counter and frequency dividers, and to enable or disable the watchdog timer function.

A reset clears BTCON to '00H'. This enables the watchdog function and selects a basic timer clock frequency of $f_{OSC}$/4096. To disable the watchdog function, you must write the signature code '1010B' to the basic timer register control bits BTCON.7–BTCON.4.

The 8-bit basic timer counter, BTCNT, can be cleared at any time during normal operation by writing a "1" to BTCON.1. To clear the frequency dividers for both the basic timer input clock and the timer 0 clock, you write a "1" to BTCON.0.

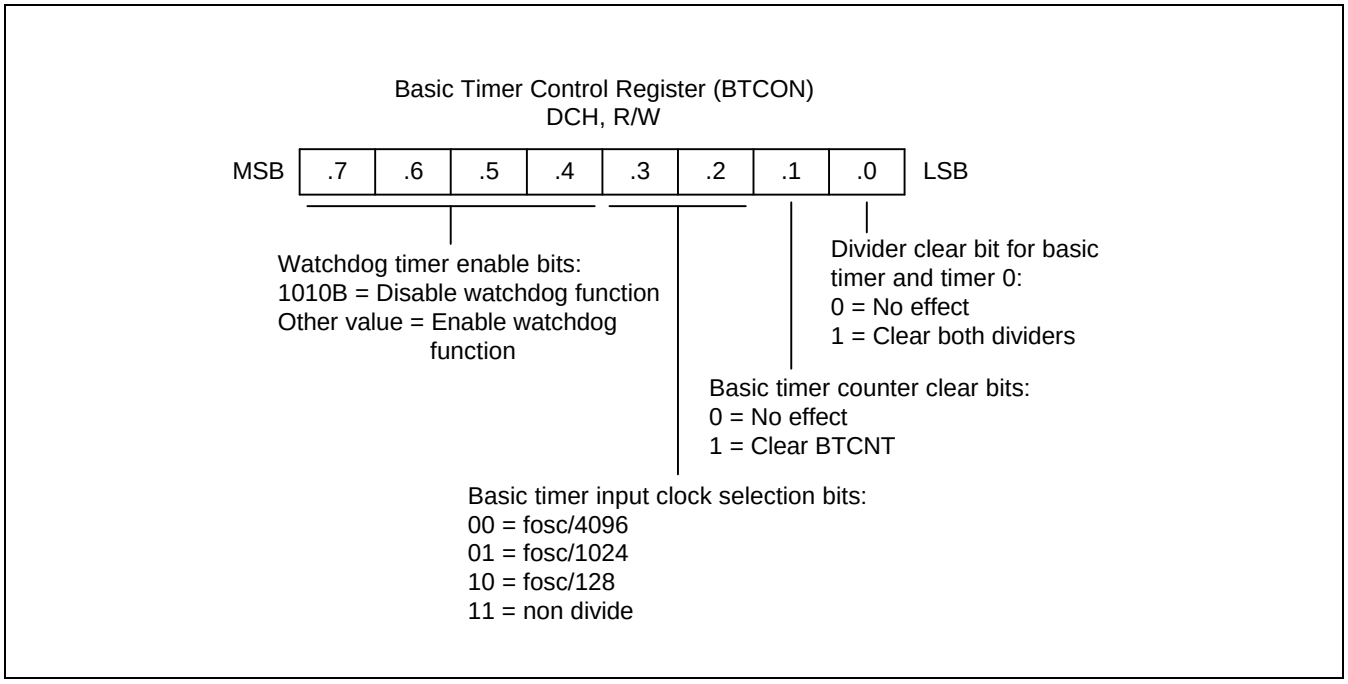



**Figure 10-1. Basic Timer Control Register (BTCON)**

## BASIC TIMER FUNCTION DESCRIPTION

### Watchdog Timer Function

You can program the basic timer overflow signal to generate a reset by setting BTCON.7–BTCON.4 to any value other than '1010B' (The '1010B' value disables the watchdog function). A reset clears BTCON to '00H', automatically enabling the watchdog timer function. A reset also selects the CPU clock (as determined by the current CLKCON register setting) divided by 4096 as the BT clock.

A reset whenever a basic timer counter overflow occurs. During normal operation, the application program must prevent the overflow, and the accompanying reset operation, from occurring. To do this, the BTCNT value must be cleared (by writing a "1" to BTCON.1) at regular intervals.

If a system malfunction occurs due to circuit noise or some other error condition, the BT counter clear operation will not be executed and a basic timer overflow will occur, initiating a reset. In other words, during normal operation, the basic timer overflow loop (a bit 7 overflow of the 8-bit basic timer counter, BTCNT) is always broken by a BTCNT clear instruction. If a malfunction does occur, a reset is triggered automatically.

### Oscillation Stabilization Interval Timer Function

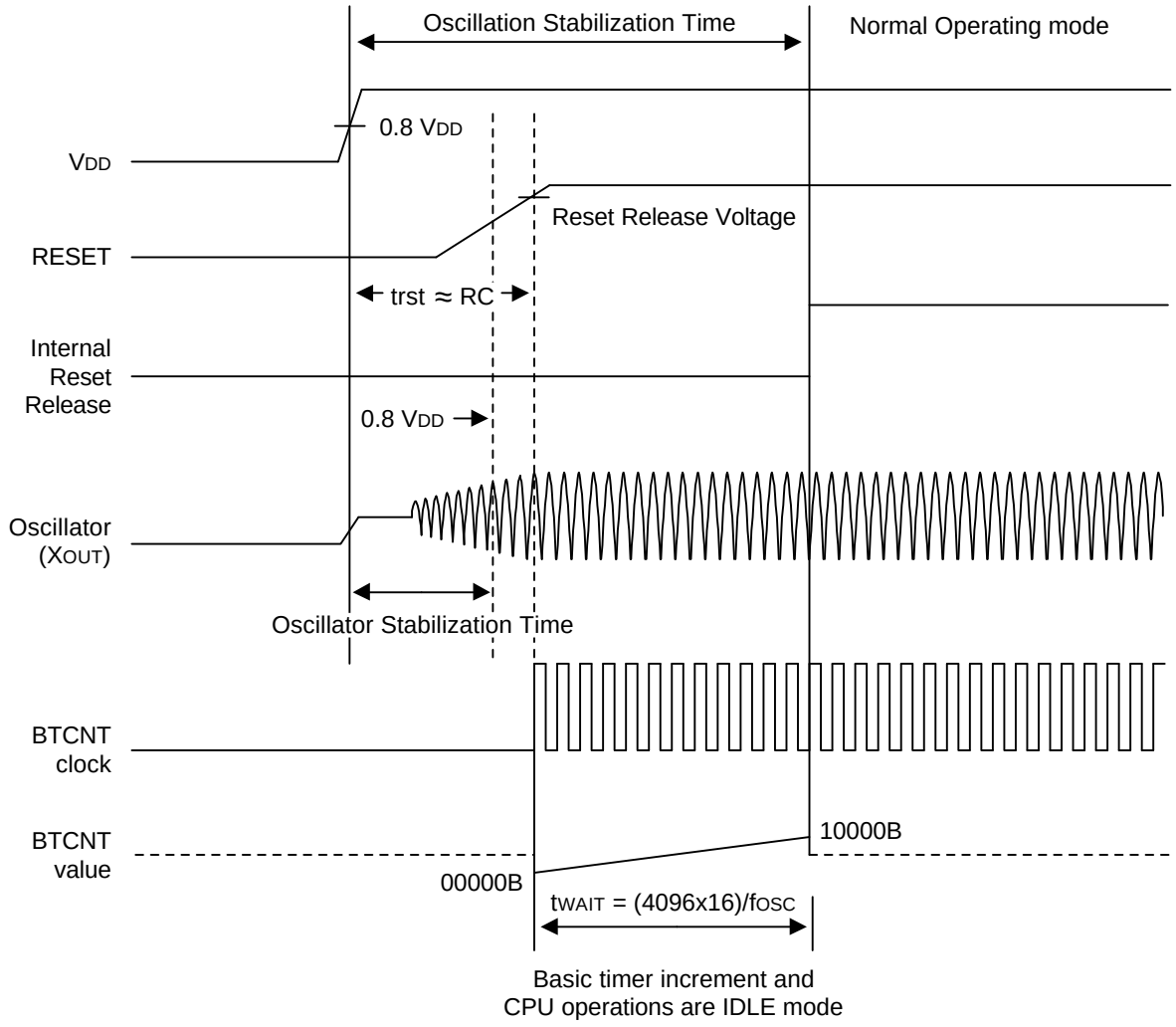
You can also use the basic timer to program a specific oscillation stabilization interval following a reset or when Stop mode has been released by an external interrupt.

In Stop mode, whenever a reset or an external interrupt occurs, the oscillator starts. The BTCNT value then starts increasing at the rate of $f_{OSC}/4096$ (for reset), or at the rate of the preset clock source (for an external interrupt). When BTCNT.4 is set, a signal is generated to indicate that the stabilization interval has elapsed and to gate the clock signal off to the CPU so that it can resume normal operation.

In summary, the following events occur when Stop mode is released:

1. During Stop mode, a power-on reset or an external interrupt occurs to trigger the Stop mode release and oscillation starts.
2. If a power-on reset occurred, the basic timer counter will increase at the rate of $f_{OSC}$ /4096. If an external interrupt is used to release Stop mode, the BTCNT value increases at the rate of the preset clock source.
3. Clock oscillation stabilization interval begins and continues until bit 4 of the basic timer counter is set.
4. When a BTCNT.4 is set, normal CPU operation resumes.

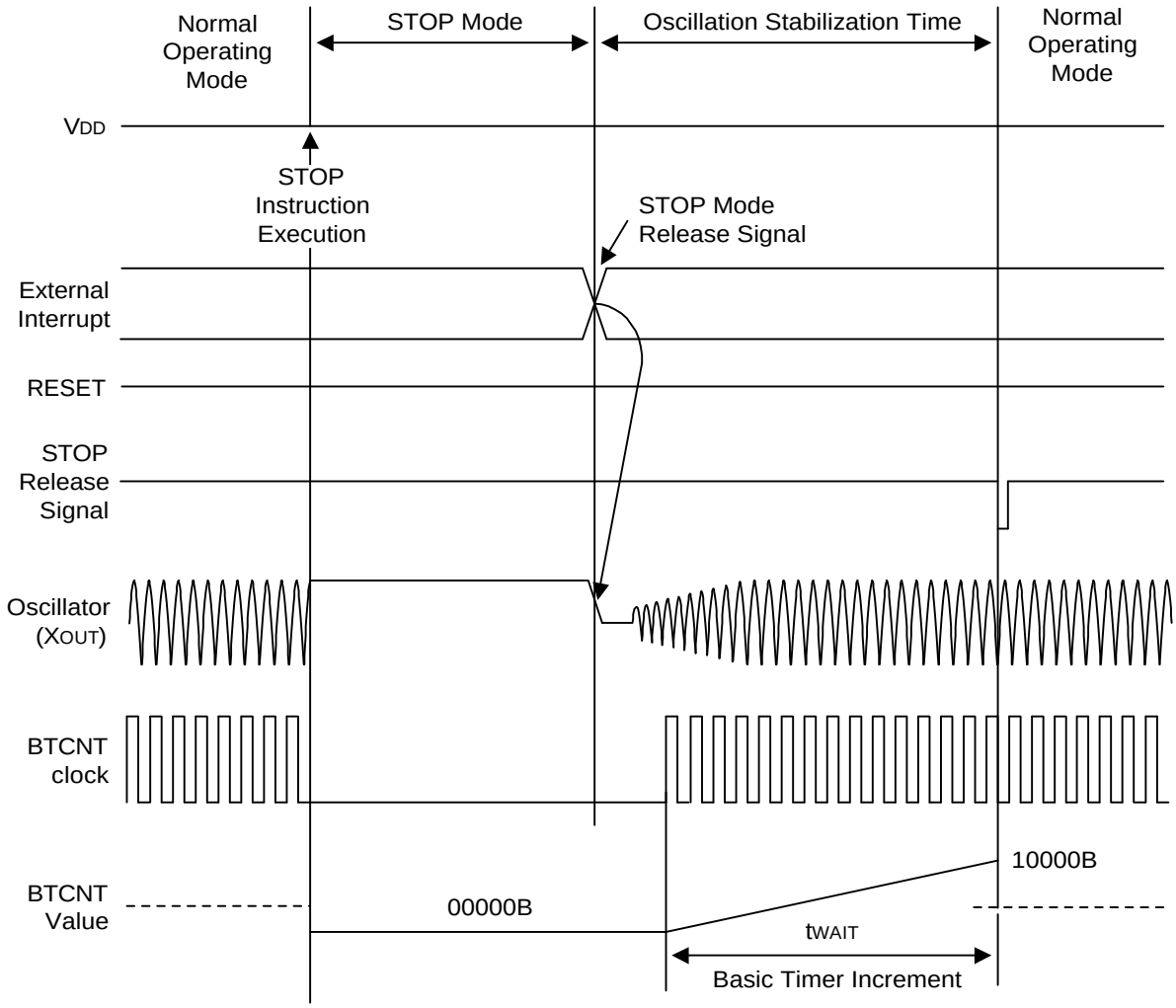
Figure 10-2 and 10-3 show the oscillation stabilization time on RESET and STOP mode release, respectively.




**NOTE:** Duration of the oscillator stabilization wait time, $t_{WAIT}$, when it is released by a Power-on-reset is 4096 x 16/$f_{OSC}$.
$t_{RST} \approx RC$ (R is external resistor and C is on chip capacitor)

**Figure 10-2. Oscillation Stabilization Time on RESET Pin Used**

**NOTE:** See Figure 14-3. For LVD Reset




**NOTE:** Duration of the oscillator stabilzation wait time, tWAIT, it is released by an interrupt is determined by the setting in basic timer control register, BTCON.

| BTCON.3 | BTCON.2 | tWAIT | tWAIT (When fosc is 6 MHz) |
|---|---|---|---|
| 0 | 0 | (4096 x 16)/fosc | 10.92 ms |
| 0 | 1 | (1024 x 16)/fosc | 2.7 ms |
| 1 | 0 | (128 x 16)/fosc | 0.341 ms |
| 1 | 1 | Invalid setting | — |

**Figure 10-3. Oscillation Stabilization Time on STOP Mode Release**

### TIMER 0 CONTROL REGISTER (T0CON)

T0CON is located at address D2H, and is read/write addressable.

A reset clears T0CON to '00H'. This sets timer 0 to normal interval match mode, selects an input clock frequency of $f_{OSC}$/4096, and disables the timer 0 overflow interrupt and match interrupt. You can clear the timer 0 counter at any time during normal operation by writing a "1" to T0CON.3.

The timer 0 overflow interrupt can be enabled by writing a "1" to T0CON.1. When a timer 0 overflow interrupt occurs and is serviced by the CPU, the pending condition must be cleared by software by writing a "0" to the timer 0 interrupt pending bit, T0CON.0.

To enable the timer 0 match interrupt, you must write T0CON.1 to "1". To detect an interrupt pending condition, the application program polls T0CON.0. When a "1" is detected, a timer 0 match/ capture interrupt is pending. When the interrupt request has been serviced, the pending condition must be cleared by software by writing a "0" to the timer 0 interrupt pending bit, T0CON.0.

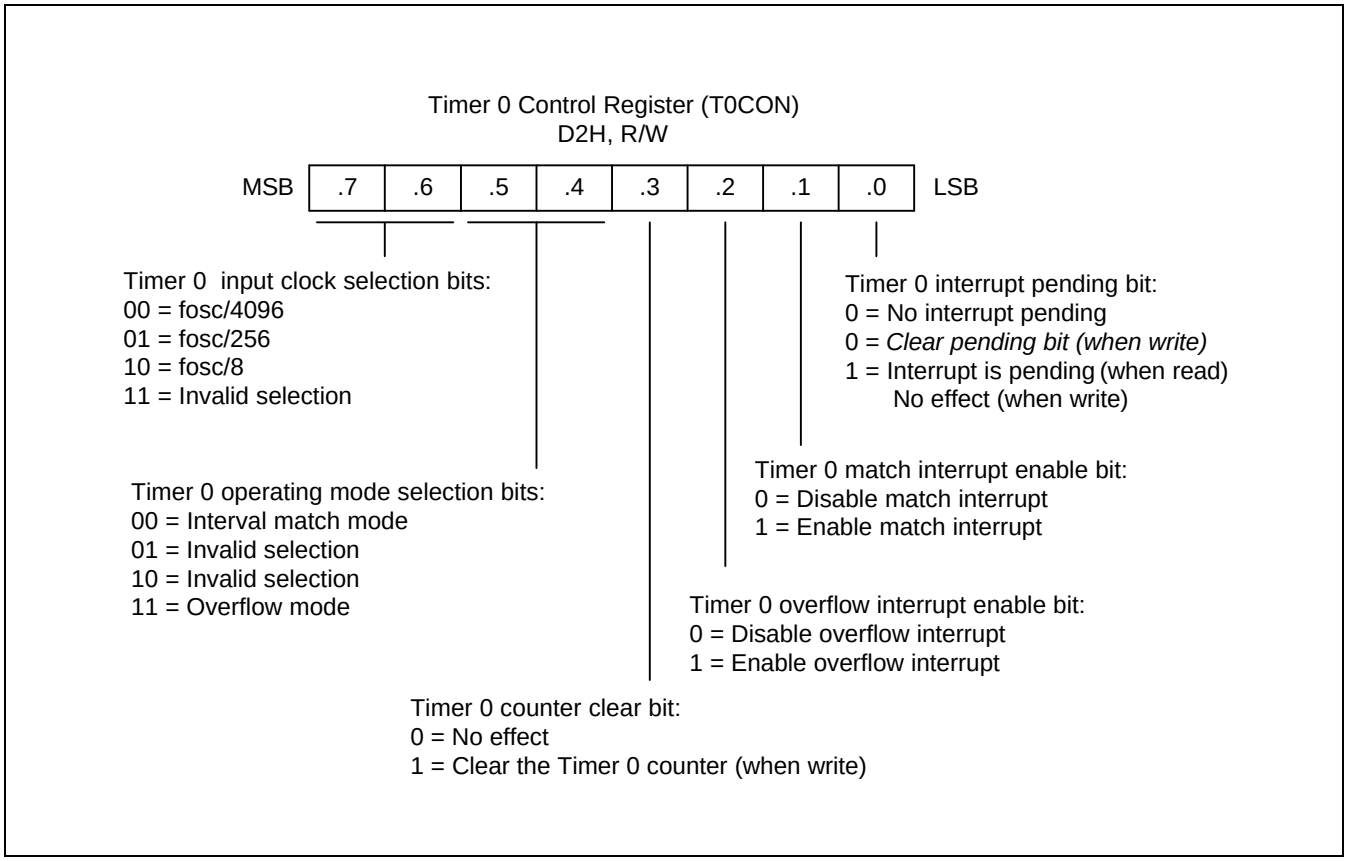



**Figure 10-4. Timer 0 Control Register (T0CON)**

## TIMER 0 FUNCTION DESCRIPTION

### Interval Match Mode

In interval match mode, a match signal is generated when the counter value is identical to the value written to the T0 reference data register, T0DATA. The match signal generates a timer 0 match interrupt and then clears the counter. If for example, you write the value '10H' to T0DATA, the counter will increment until it reaches '10H'. At this point, the T0 match interrupt is generated, the counter value is reset and counting resumes.

### Overflow Mode

In overflow mode, a overflow signal is generated regardless of the value written to the T0 reference data register when the counter value is overflowed. The overflow signal generates a timer 0 overflow interrupt and then T0 counter is cleared.

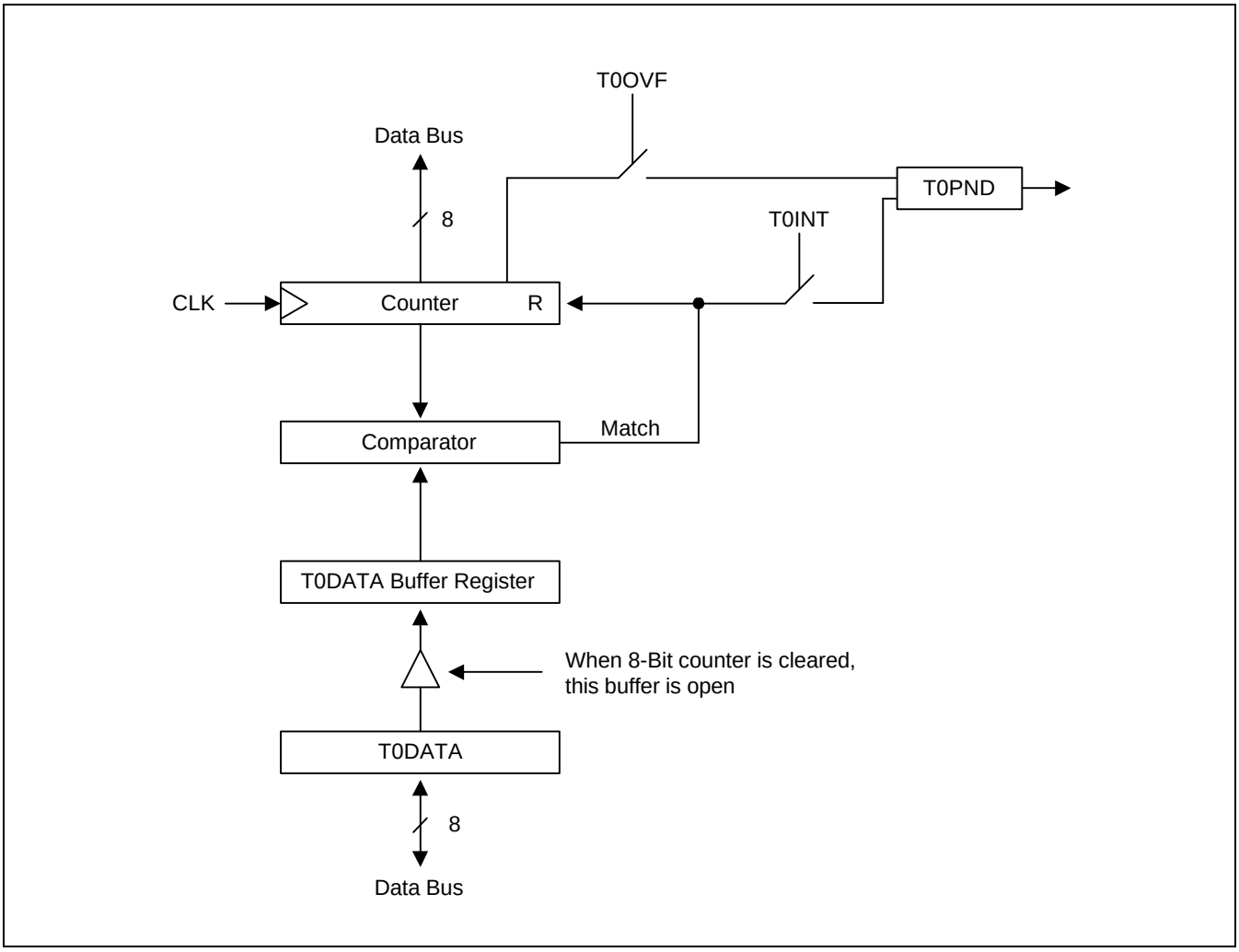



**Figure 10-5. Simplified Timer 0 Function Diagram: Interval Timer Mode**

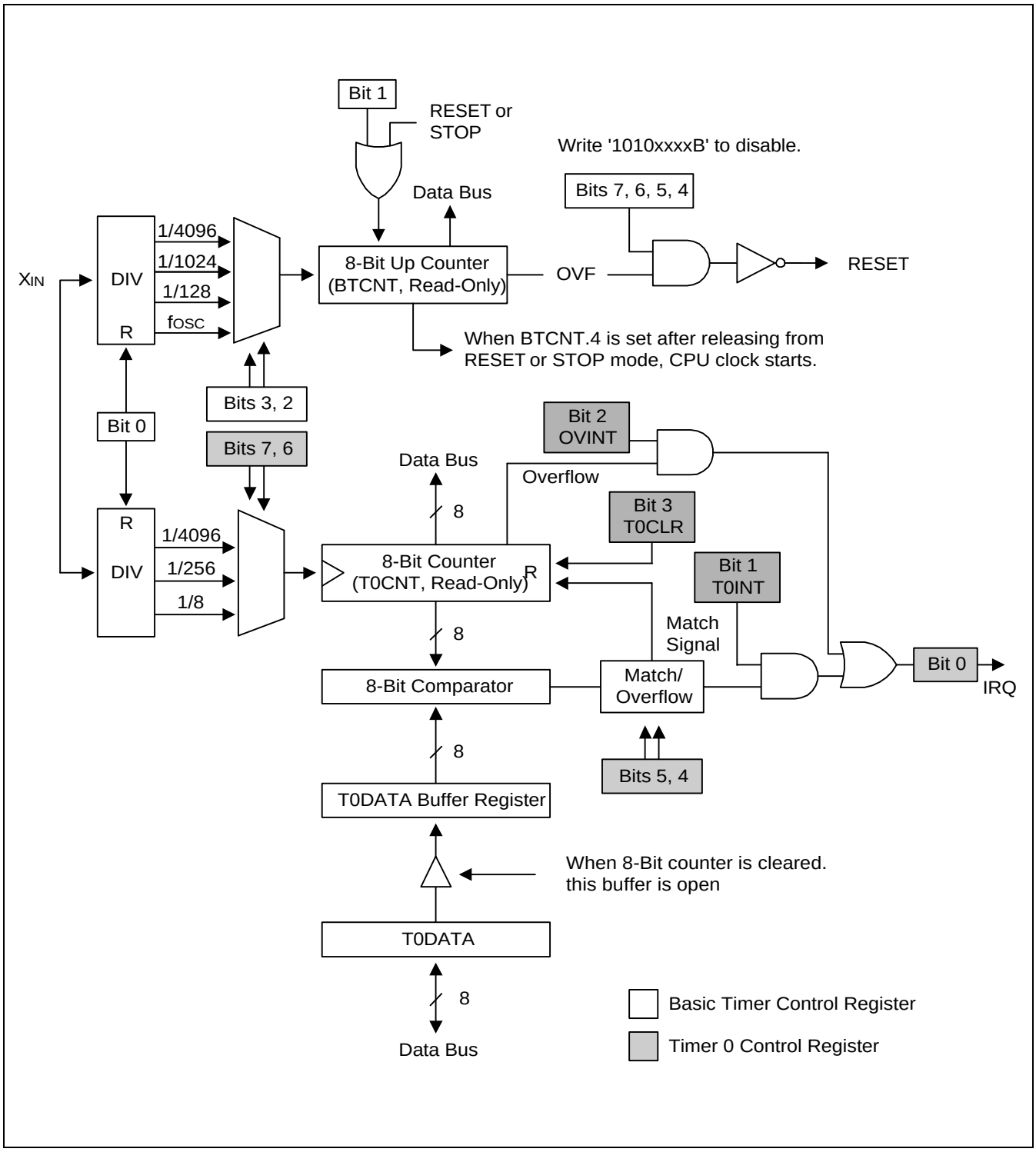



**Figure 10-6. Basic Timer and Timer 0 Block Diagram**

# 11 UNIVERSAL SERIAL BUS

## OVERVIEW

Universal Serial Bus (USB) is a communication architecture that supports data transfer between a host computer and a wide range of PC peripherals. USB is actually a cable bus in which the peripherals share its bandwidth through a host scheduled token based protocol.

The USB module in S3C9654/C9658/P9658 is designed to serve at a low speed transfer rate (1.5 Mbs) USB device as described in the Universal Serial Bus Specification Revision 1.0. S3C9654/C9658/P9658 can be briefly describe as a microcontroller with SAM 87RCRI core with an on-chip USB peripheral as can be seen in figure 11-1.

The S3C9654/C9658/P9658 comes equipped with Serial Interface Engine (SIE), which handles the communication protocol of the USB. The S3C9654/C9658/P9658 supports the following control logic: packet decoding/generation, CRC generation/checking, NRZI encoding/decoding, Sync detection, EOP (end of packet) detection and bit stuffing.

S3C9654/C9658/P9658 supports two types of data transfers; control and interrupt. Two endpoints are used in this device; Endpoint 0 and Endpoint 1. Please refer to the USB specification revision 1.0 for detail description of USB.

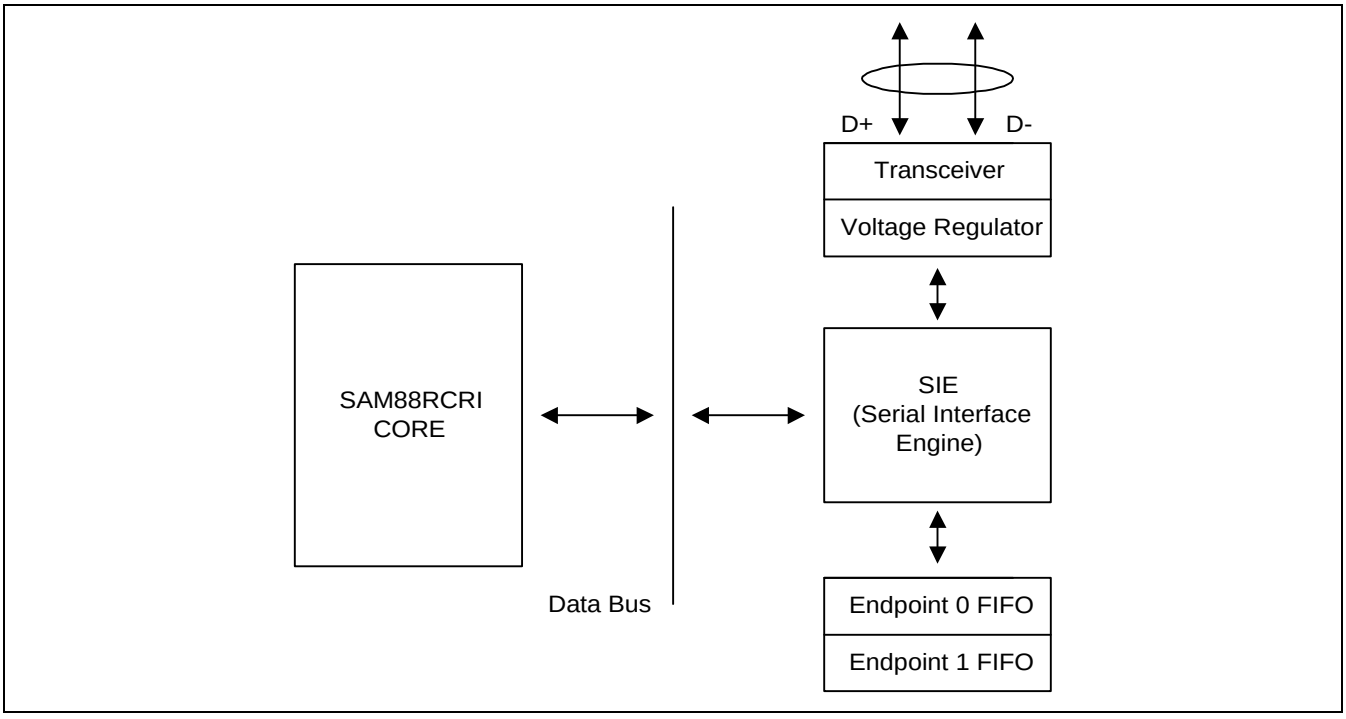



**Figure 11-1. USB Peripheral Interface**

### Serial Bus Interface Engine (SIE)

The Serial Interface Engine interfaces to the USB serial data and handles, deserialization/serialization of data, NRZI encoding/decoding, clock extraction, CRC generation and checking, bit stuffing and other specifications pertaining to the USB protocol such as handling inter packet time out and PID decoding.

### Control Logic

The USB control logic manages data movements between the CPU and the transceiver by manipulating the transceiver and the endpoint register. This includes both transmit and receive operations on the USB. The logic contains byte count buffers for transmit operations that load the active transmit endpoint's byte count and use this to determine the number of bytes to transfer. The same buffer is used for receive transactions to count the number of bytes received and transfer that number to the receive endpoint's byte count register at the end of the transaction.

The control logic in S3C9654/C9658/P9658, when transmitting, manages parallel to serial conversion, packet generation, CRC generation, NRZI encoding and bit stuffing.

When receiving, the control logic in S3C9654/C9658/P9658 handles Sync detection, packet decoding, EOP (end of packet) detection, bit (un)stuffing, NRZI decoding, CRC checking and serial to parallel conversion

### Bus Protocol

All bus transactions involve the transmission of packets. S3C9654/C9658/P9658 supports three packet types; Token, Data and Handshake. Each transaction starts when the host controller sends a Token Packet to the USB device. The Token packets are generated by the USB host and decoded by the USB device. A Token Packet includes the type description, direction of the transaction, USB device address and the endpoint number.

Data and Handshake packets are both decoded and generated by the USB device. In any transaction, the data is transferred from the host to a device or from a device to the host. The transaction source then sends a Data Packet or indicates that it has no data to transfer. The destination then responds with a Handshake Packet indicating whether the transfer was successful.

### Data Transfer Types

USB data transfer occurs between the host software and a specific endpoint on the USB device. An endpoint supports a specific type of data transfer. The S3C9654/C9658/P9658 supports two data transfer endpoints: control and interrupt.

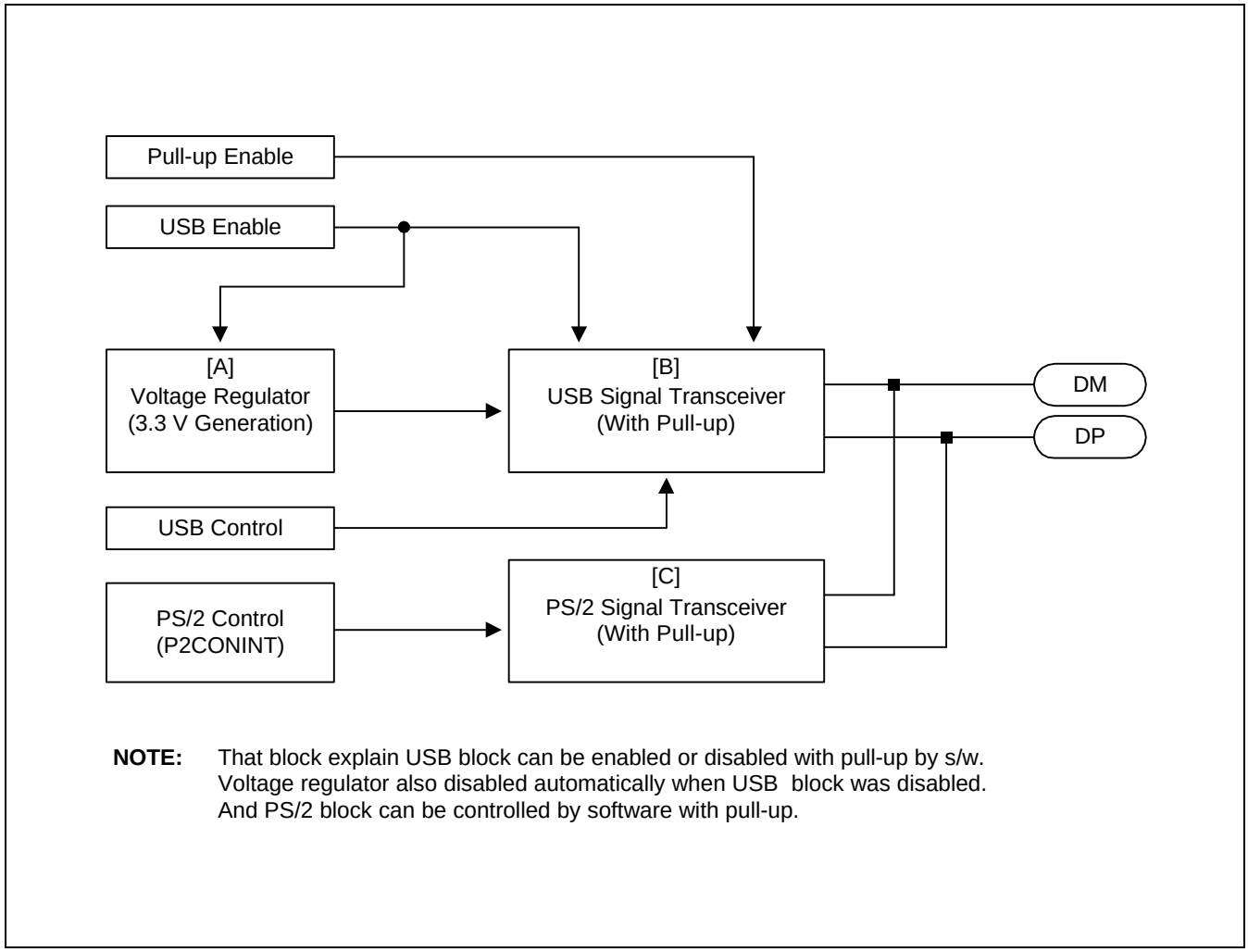
Control transfer configures and assigns an address to the device when detected. Control transfer also supports status transaction, returning status information from device to host.

Interrupt transfer refers to a small, spontaneous data transfer from USB device to host.

### Endpoints

Communication flows between the host software and the endpoints on the USB device. Each endpoint on a device has an identifier number. In addition to the endpoint number, each endpoint supports a specific transfer type. S3C9654/C9658/P9658 supports two endpoints: Endpoint 0 supports control transfer, and Endpoint 1 supports interrupt transfer.

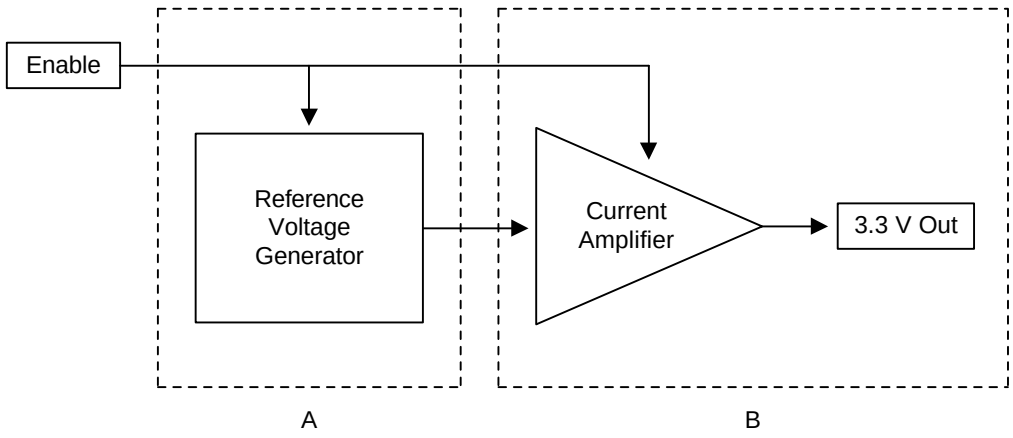
## STRUCTURE OF USB AND PS/2 COMBINATIONAL PORT




**NOTE:** That block explain USB block can be enabled or disabled with pull-up by s/w. Voltage regulator also disabled automatically when USB block was disabled. And PS/2 block can be controlled by software with pull-up.

**Figure 11-2. Block Diagram of USB and PS/2 Transceiver**

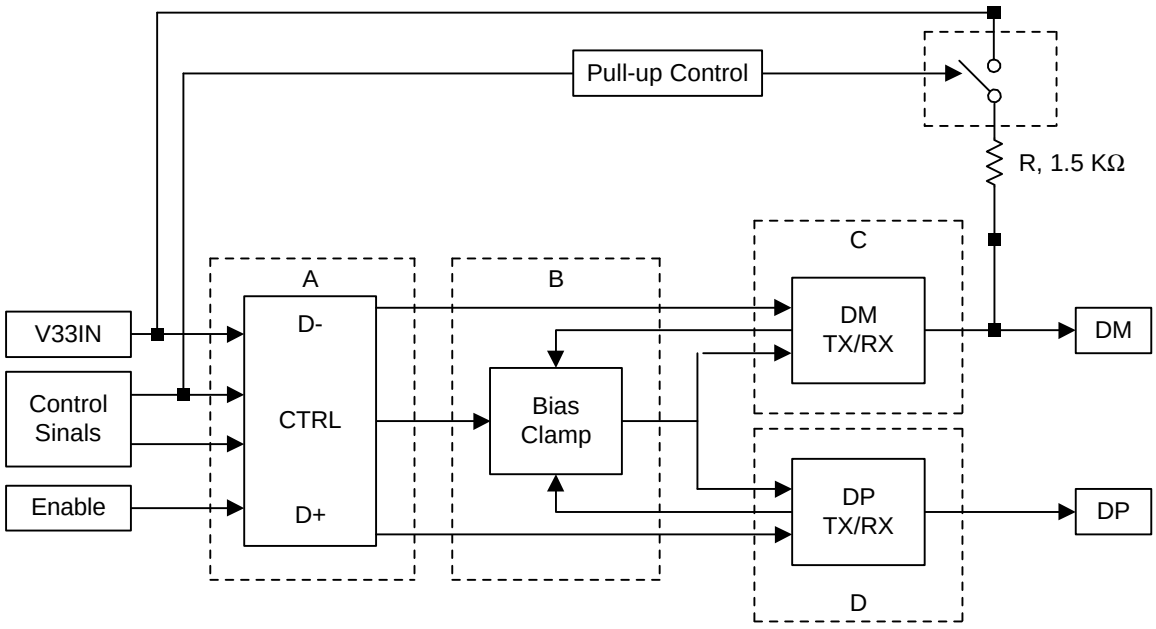
## STRUCTURE OF VOLTAGE REGULATOR




**NOTE:** This blcok can give a explanation how it can be controlled automatically.
When the 3.3 voltage regulator be enable by software, voltage regulator will operating to cover fluctuation of the line load, sometimes the line is not stabled and the driving ability will be dropped.
As it operating in the normal stage without any peak, power will be supplied with 8 mA, and when the operating.
Current consumption go to peak, it was designed to cover by 50 mA.
It means any kind of load problem will be compensated with above design.

**Figure 11-3. Block Diagram of Voltage Regulator**

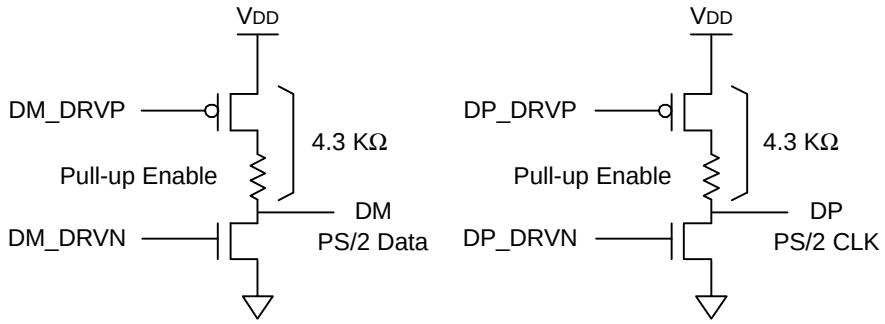
## STRUCTURE OF USB SIGNAL TRANSMITTER




**NOTE:** We didn't used the by-pass capacitor on the 3.3 V out, since the 3.3 V regulator and clamp circuit will give a solution through the feedback.
USB block was designed to cover the line load, the typical value designed is 300 pF (max: 650 pF).
The calmp block operating after it detect the voltage variation
(actually the current fluctuation will be feedback into voltage variation, di/dt to dv/dt variation.
Bias control the slope.
Control signals means NRZI, EOP, XCON, IN/OUT.
Enable is for the Tx, Rx.
Internal pull-up resistor will be 1.5 KΩ

**Figure 11-4. Block Diagram of USB Signal Transceiver**

## STRUCTURE OF PS/2 SIGNAL TRANSMITTER




**NOTE:** It explain the PS/2 block.
The pull-up resistor value will be 4.3 kΩ ± 20 % (Vport = 0 V)
This block can be controlled with pull-up resistor
and it was designed with totally different from usb.

**Figure 11-5. Block Diagram of PS/2 Signal Transmitter**

## USB FUNCTION ADDRESS REGISTER (FADDR)

This register holds the USB address assigned by the host computer. USBADDR is located at address F0H and is read/write addressable.

Bit7 Not used

Bit6–0 **FADDR:** MCU updates this register once it decodes a SET_ADDRESS command. MCU must write this register before it clears OUT_PKT_RDY (bit0) and sets DATA_END (bit3) in the EP0STU register. The function controller use this register's value to decode USB Token packet address. At reset, if the device is not yet configured the value is reset to 0.

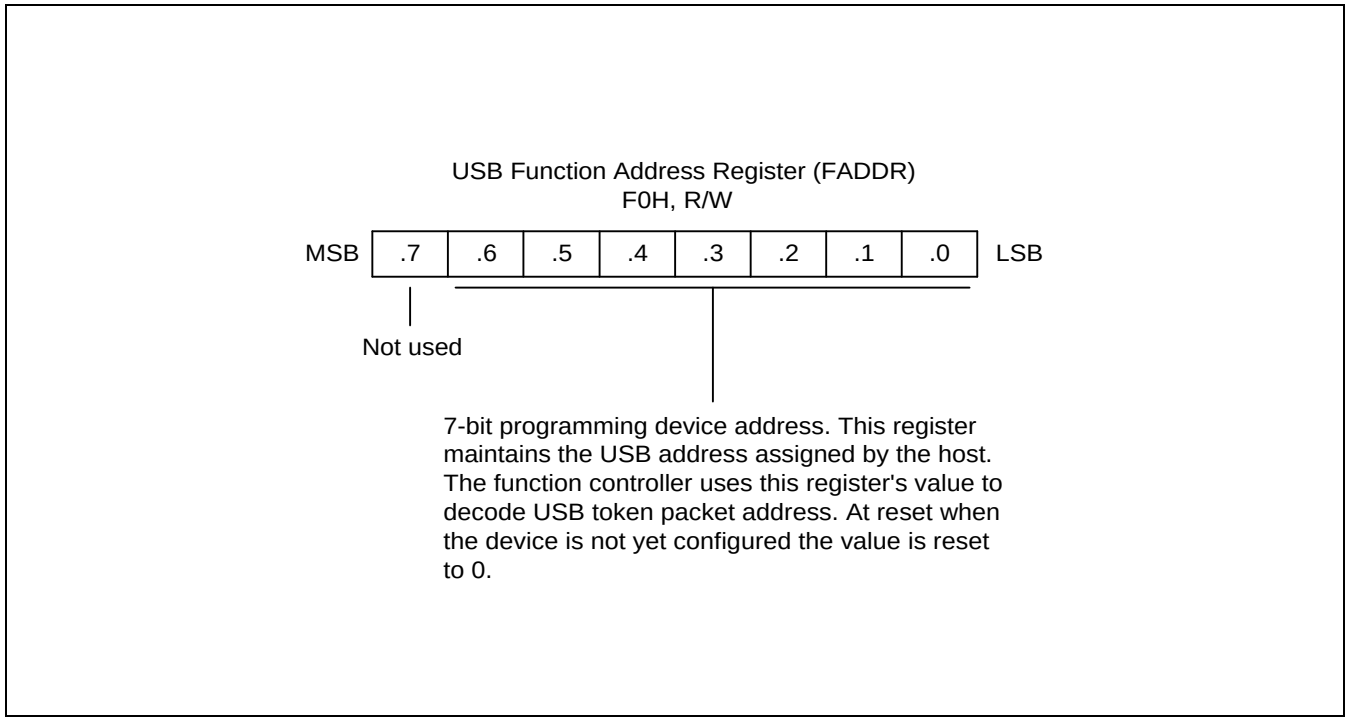



**Figure 11-6. USB Function Address Register (FADDR)**

## CONTROL ENDPOINT STATUS REGISTER (EP0CSR)

EP0CSR register controls Endpoint 0 (Control Endpoint), and also holds status bits for Endpoint 0. EP0CSR is located at F1H and is read/write addressable.

Bit7 **CLEAR_SETUP_END:** MCU writes “1” to this bit to clear SETUP_END bit (bit4). This bit is automatically cleared after writing "1" by USB block.

Bit6 **CLEAR_OUT_PKT_RDY:** MCU writes “1” to this bit to clear OUT_PKT_RDY bit (bit0). This bit is automatically cleared after writing "1" by USB block.

Bit5 **SEND_STALL:** MCU writes “1” to this bit to send STALL signal to the Host, at the same time it clears OUT_PKT_RDY (bit0), if it decodes an invalid token. USB issues a STALL Handshake to the current control transfer. This bit gets cleared once a STALL Handshake is issued to the current control transfer.

Bit4 **SETUP_END:** USB sets this bit, when a control transfer ends before DATA_END bit (bit3) is set. MCU clears this bit, by writing a “1” to SERVICED_SETUP_END bit (bit7). When USB sets this bit, an interrupt is generated to MCU. When such condition occurs, USB flushes the FIFO, and invalidates MCU’s access to FIFO.

Bit3 **DATA_END:** MCU sets this bit:

— After loading the last packet of data into the FIFO, and at the same time IN_PKT_RDY bit is set.
— While it clears OUT_PKT_RDY bit after unloading the last packet of data.
— For a zero length data phase, when it clears OUT_PKT_RDY bit, and sets IN_PKT_RDY bit.

Bit2 **SENT_STALL:** USB sets this bit, if a control transaction has ended due to a protocol violation. An interrupt is generated when this bit gets set. MCU clears this bit to end the STALL condition.

Bit1 **IN_PKT_RDY:** MCU sets this bit, after writing a packet of data into Endpoint 0 FIFO. USB clears this bit, once the packet has been successfully sent to the host. An interrupt is generated when USB clears this bit so that MCU can load the next packet. For a zero length data phase, MCU sets IN_PKT_RDY bit and DATA_END bit at the same time.

Bit0 **OUT_PKT_RDY:** USB sets this bit, once a valid token is written to FIFO. An interrupt is generated, when USB sets this bit. MCU clears this bit by writing "1” to SERVICED_OUT_PKT_RDY bit.

**NOTES:**
1. In control transfer case, where there is no data phase, MCU after unloading the setup token, sets IN_PKT_RDY, and DATA_END at the same time it clears OUT_PKT_RDY for the setup token.
2. When SETUP_END bit is set, OUT_PKT_RDY bit may also be set. This happens when the current transfer has ended, and a new control transfer is received before MCU can service the interrupt. In such case, MCU should first clear SETUP_END bit, and then start servicing the new control transfer.

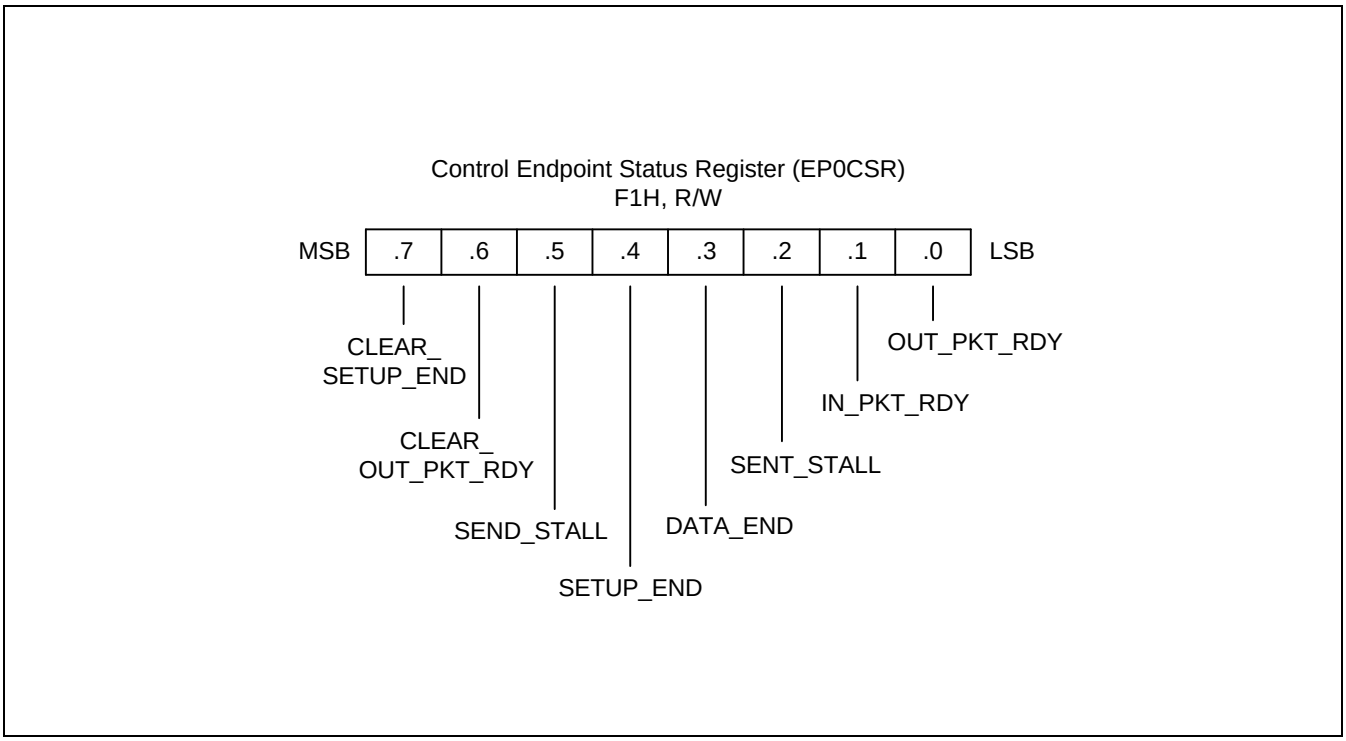



**Figure 11-7. Control Endpoint Status Register (EP0CSR)**

### INTERRUPT ENDPOINT STATUS REGISTER (EP1CSR)

EP1CSR is the control register for Endpoint 1, Interrupt Endpoint. This register is located at address F2H and is read/write addressable.

Bit7 **CLR_DATA_TOGGLE:** MCU writes “1” to this bit to clear the data toggle sequence bit. When the MCU writes a 1 to this register, the data toggle bit is initialized to DATA0.

Bit6–3 **MAXP:** These bits indicate the maximum packet size for IN endpoint, and needs to be updated by MCU before it sets IN_PKT_RDY. Once set, the contents are valid till MCU re-writes them.

Bit2 **FLUSH_FIFO:** When MCU writes “1” to this register, the FIFO is flushed, and IN_PKT_RDY cleared. The MCU should wait for IN_PKT_RDY to be cleared for the flush to take place.

Bit1 **FORCE_STALL:** MCU writes “1” to this register to issue a STALL Handshake to USB. MCU clears this bit, to end the STALL condition.

Bit0 **IN_PKT_RDY:** MCU sets this bit, after writing a packet of data into Endpoint 1 FIFO. USB clears this bit, once the packet has been successfully sent to the Host. An interrupt is generated when USB clears this bit, so MCU can load the next packet.

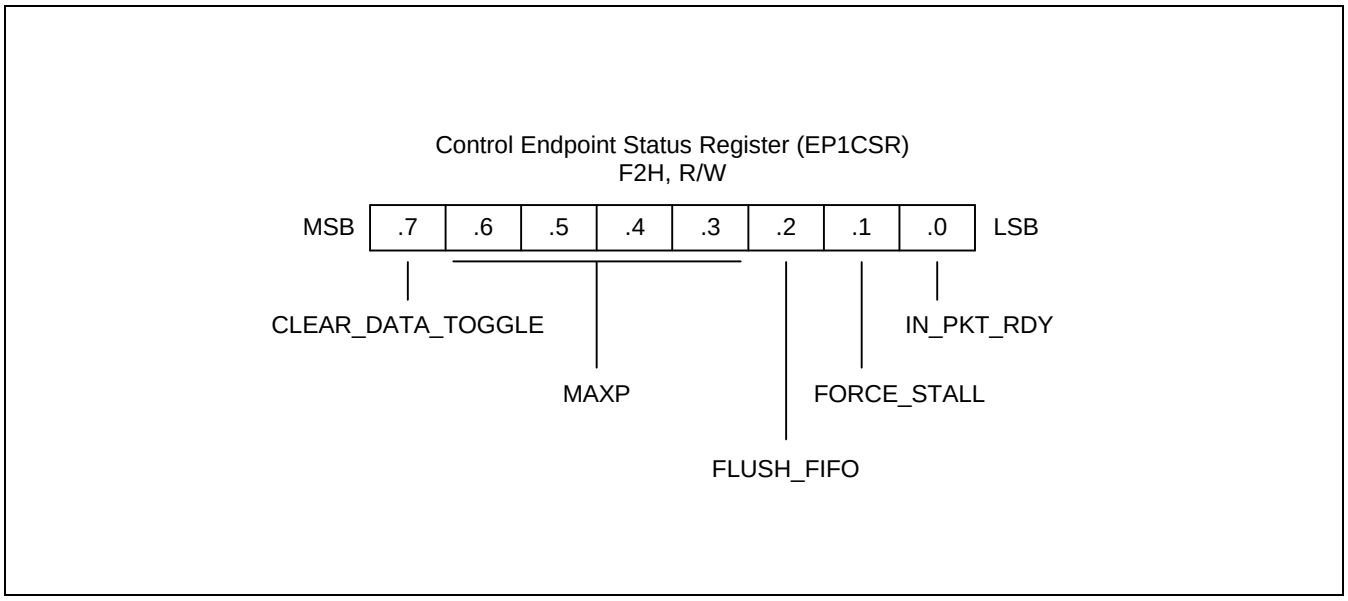



**Figure 11-8. Interrupt Endpoint Status Register (EP1CSR)**

### CONTROL ENDPOINT BYTE COUNT REGISTER (EP0BCNT)

EP0BCNT register has the number of valid bytes in Endpoint 0 FIFO. It is located at address F3H read only addressable. Once the MCU receives a OUT_PKT_RDY (Bit0 of EP0CSR) for Endpoint 0, then it can read this register to find out the number of bytes to be read from Endpoint 0 FIFO.

### CONTROL ENDPOINT FIFO REGISTER (EP0FIFO)

This register is bi-directional, 8 byte depth FIFO used to transfer Control Endpoint data. EP0FIFO is located at address F4H and is read/write addressable.

Initially, the direction of the FIFO, is from the Host to the MCU. After a setup token is received for a control transfer, that is, after MCU unload the setup token bytes, and clears OUT_PKT_RDY, the direction of FIFO is changed automatically from MCU to the Host.

### INTERRUPT ENDPOINT FIFO REGISTER (EP1FIFO)

EP1FIFO is an uni-direction 8-byte depth FIFO used to transfer data from the MCU to the Host. MCU writes data to this register, and when finished set IN_PKT_RDY. This register is located at address F5H.

### USB INTERRUPT PENDING REGISTER (USBPND)

USBPND register has the interrupt bits for endpoints and power management. *This register is cleared once read by MCU.* While any one of the bits is set, an interrupt is generated. USBPND is located at address F6H.

Bit7–4 Not used

Bit3 **RESUME_PND:** While in suspend mode, if resume signaling is received this bit gets set.

Bit2 **SUSPEND_PND:** This bit is set, when suspend signaling is received.

Bit1 **ENDPT1_PND:** This bit is set, when Endpoint 1 needs to be serviced.

Bit0 **ENDPT0_PND:** This bit is set, when Endpoint 0 needs to be serviced. It is set under any one of the following conditions:

- OUT_PKT_RDY is set.
- IN_PKT_RDY gets cleared.
- SENT_STALL gets set.
- DATA_END gets cleared.
- SETUP_END gets set.

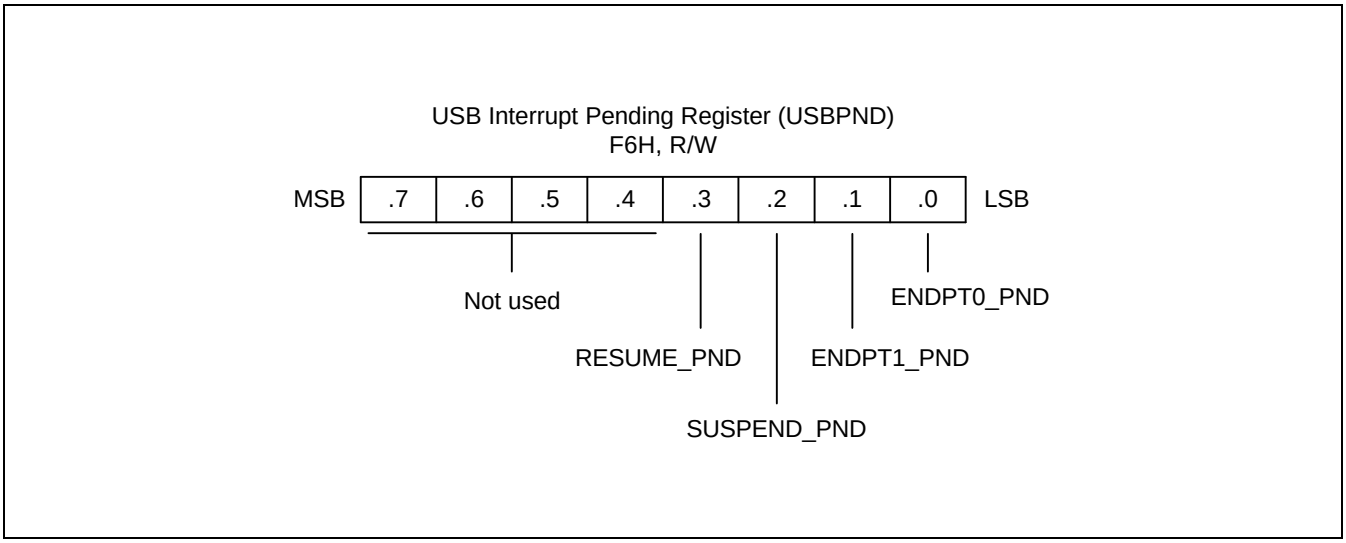



**Figure 11-9. USB Interrupt Pending Register (USBPND)**

## USB INTERRUPT ENABLE REGISTER (USBINT)

USBINT is located at address F7H and is read/write addressable. This register serves as an interrupt mask register. If the corresponding bit = 1 then the respective interrupt is enabled.

By default, all interrupts except suspend interrupt is enabled. Interrupt enables bits for suspend and resume is combined into a single bit (bit 2).

Bit7–3 Not used

Bit2 **ENABLE_SUSPEND_RESUME_INT:**
1 Enable SUSPEND and RESUME INTERRUPT
0 Disable SUSPEND and RESUME INTERRUPT (default)

Bit1 **ENABLE_ENDPT1_INT:**
1 Enable ENDPOINT 1 INTERRUPT (default)
0 Disable ENDPOINT 1 INTERRUPT

Bit0 **ENABLE_ENDPT0_INT:**
1 Enable ENDPOINT 0 INTERRUPT (default)
0 Disable ENDPOINT 0 INTERRUPT

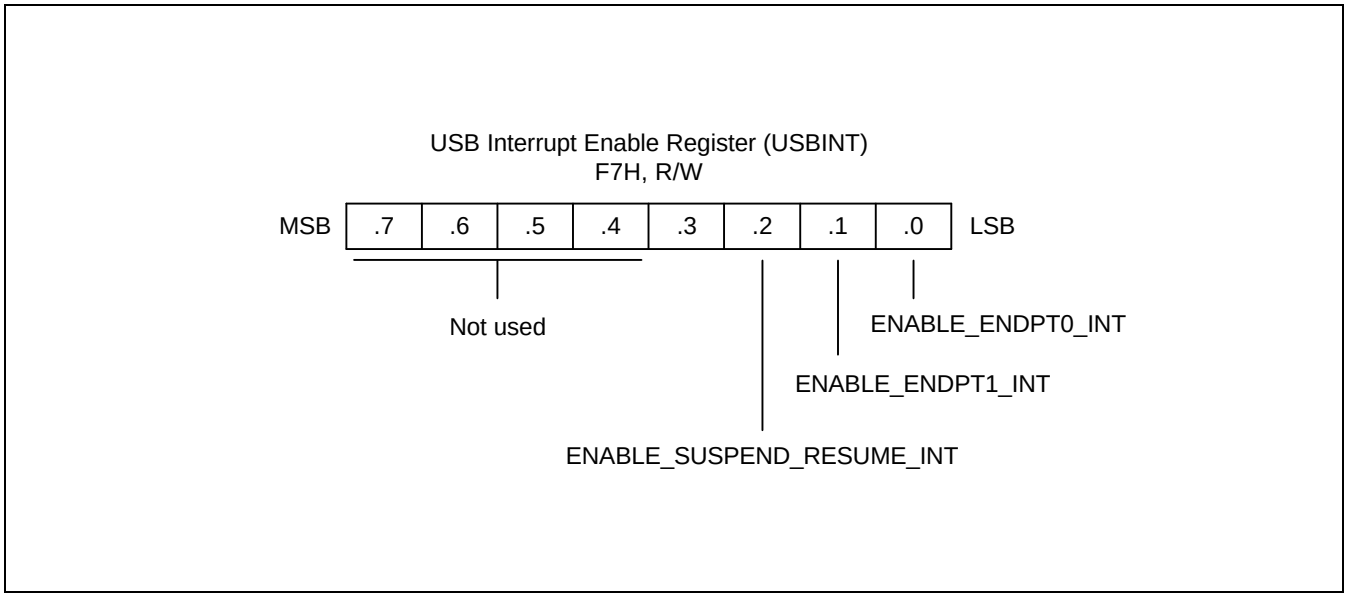



**Figure 11-10. USB Interrupt Enable Register (USBINT)**

### USB POWER MANAGEMENT REGISTER (PWRMGR)

PWRMGR register interacts with the Host's power management system to execute system power events such as SUSPEND or RESUME. This register is located at address F8H and is read/write addressable.

Bit7–2 **RESERVED:** The value read from this bit is zero.

Bit1 **SEND_RESUME:** While in SUSPEND state, if the MCU wants to initiate RESUME, it writes "1" to this register for 10ms (maximum of 15ms), and clears this register. In SUSPEND mode if this bit reads "1", USB generates RESUME signaling.

Bit0 **SUSPEND_STATE:** Suspend state is set when the MCU sets suspend interrupt. This bit is cleared automatically when:

— MCU writes "0" to SEND_RESUME bit to end the RESUME signaling (after SEND_RESUME is set for 10ms).
— MCU receives RESUMES signaling from the Host while in SUSPEND mode.

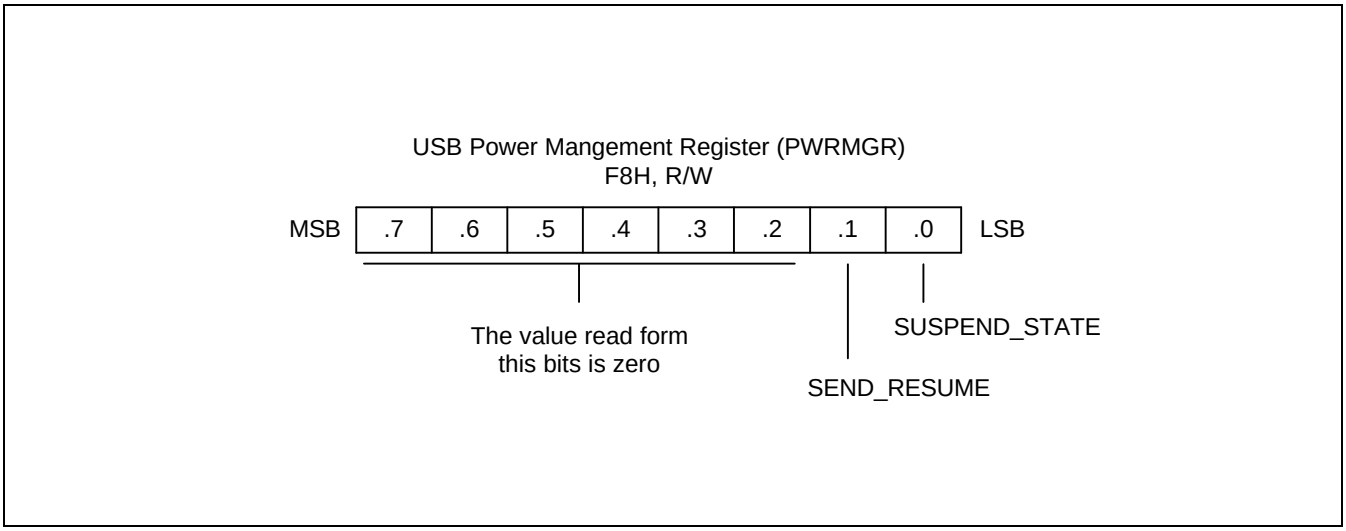



**Figure 11-11. USB Power Management Register (PWRMGR)**

### USB RESET REGISTER (USBRST)

USBRST register receives a reset signal from the Host. This register is located at address FFH and is read/write addressable.

Bit7–1 Not used

Bit0 **USBRST:** This bit is set when the Host issues an USB reset signal.

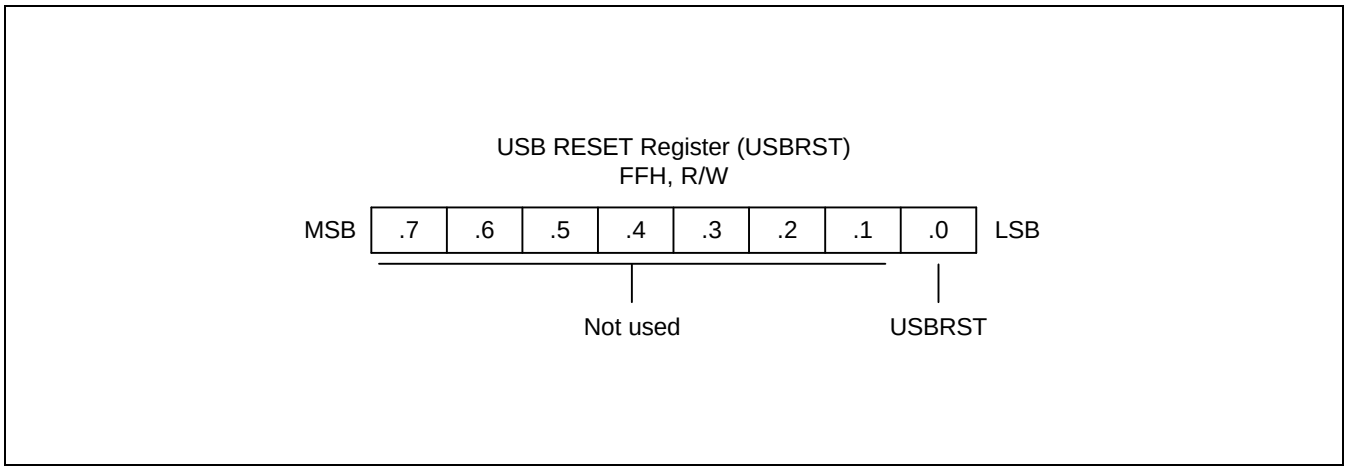



**Figure 11-12. USB RESET Register (USBRST)**

# 12 COMPARATOR

## OVERVIEW

P1.0–P1.5 can be used as a analog input port for a comparator. The reference voltage for the 6-channel comparator can be supplied either internally or externally at P1.5. When an internal reference voltage is used, six channels (P1.0–P1.5) are used for analog inputs and the internal reference voltage is varied in 32 levels. If an external reference voltage is input at P1.5, the other three pins (P1.0–P1.4) in port x are used analog input. Unused port x pins should be connected to $V_{DD}$ or $V_{SS}$ for current saving.

When a conversion is completed, the result is saved in the comparison result register CDATA. The initial values of the CDATA are undefined and the comparator operation is disabled by a RESET.

— Analog comparator
— Internal reference voltage generator (5-bit resolution)
— External reference voltage source at P1.5
— Comparator mode register (CCON)
— Four multiplexed analog data input pins (CIN0–CIN5)
— 6–channel conversion data result register (CDATA)
— 6–bit digital input port (alternatively, I/O port)
— Internal reference voltage is varied in 32 levels with hysteresis.

## FUNCTION DESCRIPTION

The comparator compares analog voltage input at CIN0–CIN5 with an external or internal reference voltage ($V_{REF}$) that is selected by CCON register. The result is written to the comparison result register CDATA at address E5H. The comparison result is calculated as follows.

If “1” Analog input voltage ≥ $V_{REF}$ + 100 mV

If “0” Analog input voltage ≤ $V_{REF}$ – 100 mV

To obtain a comparison result, the data must be read out from the CDATA register after $V_{REF}$ is updated by changing the CCON value after a conversion time has elapsed.

## COMPARATOR CONTROL REGISTER (CCON)

The comparator control register CCON is an 8–bit register that is used to set the operation mode of the comparator. To initiate a comparison procedure, you write the reference voltage selection data in the comparator control register CCON and set the comparison start of enable bit, CCON.7.

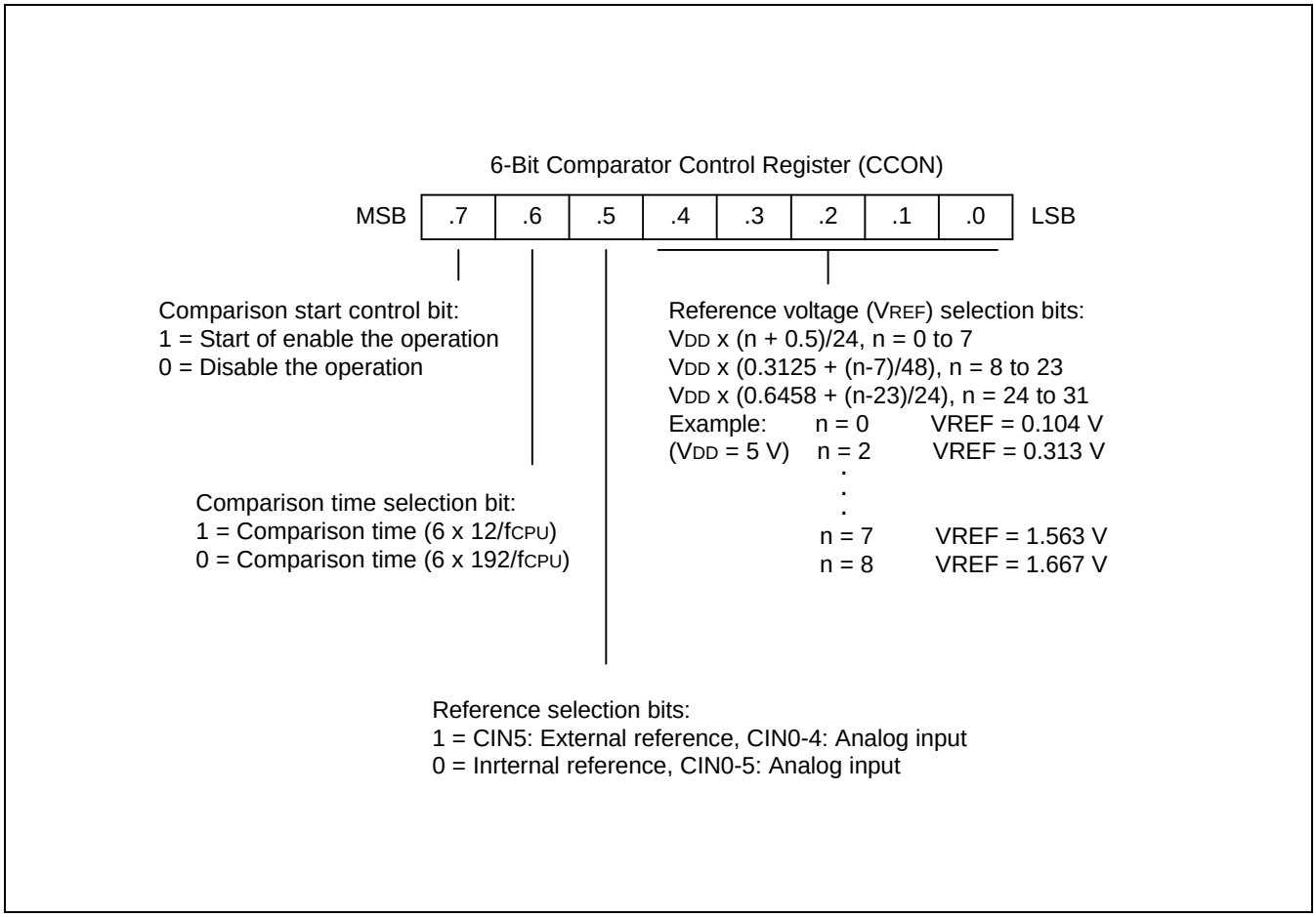



**Figure 12-1. Comparison Control Register (CCON)**

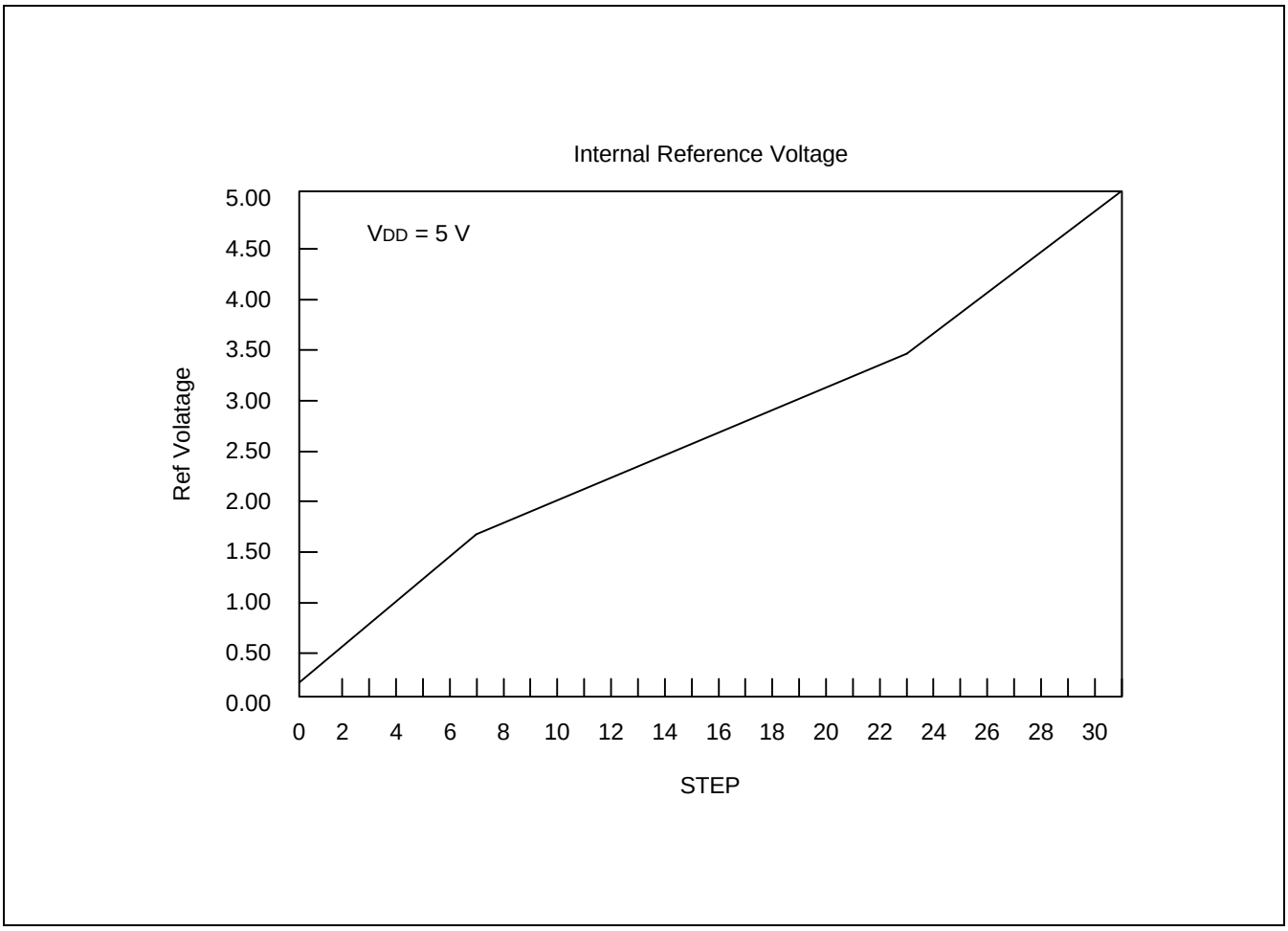



**Figure 12-2. Internal Reference Voltage (Hysteresis)**

## BLOCK DIAGRAM

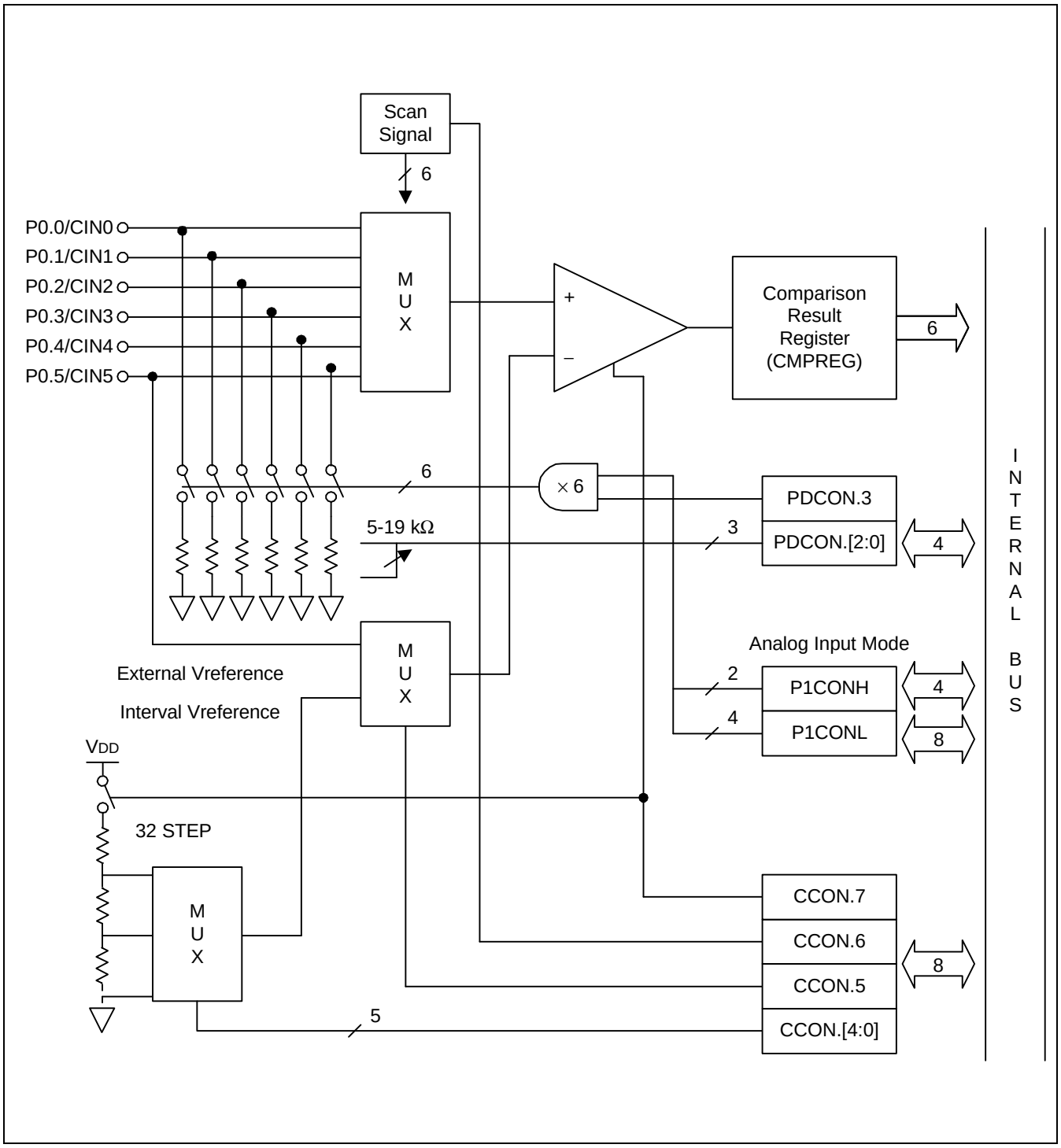


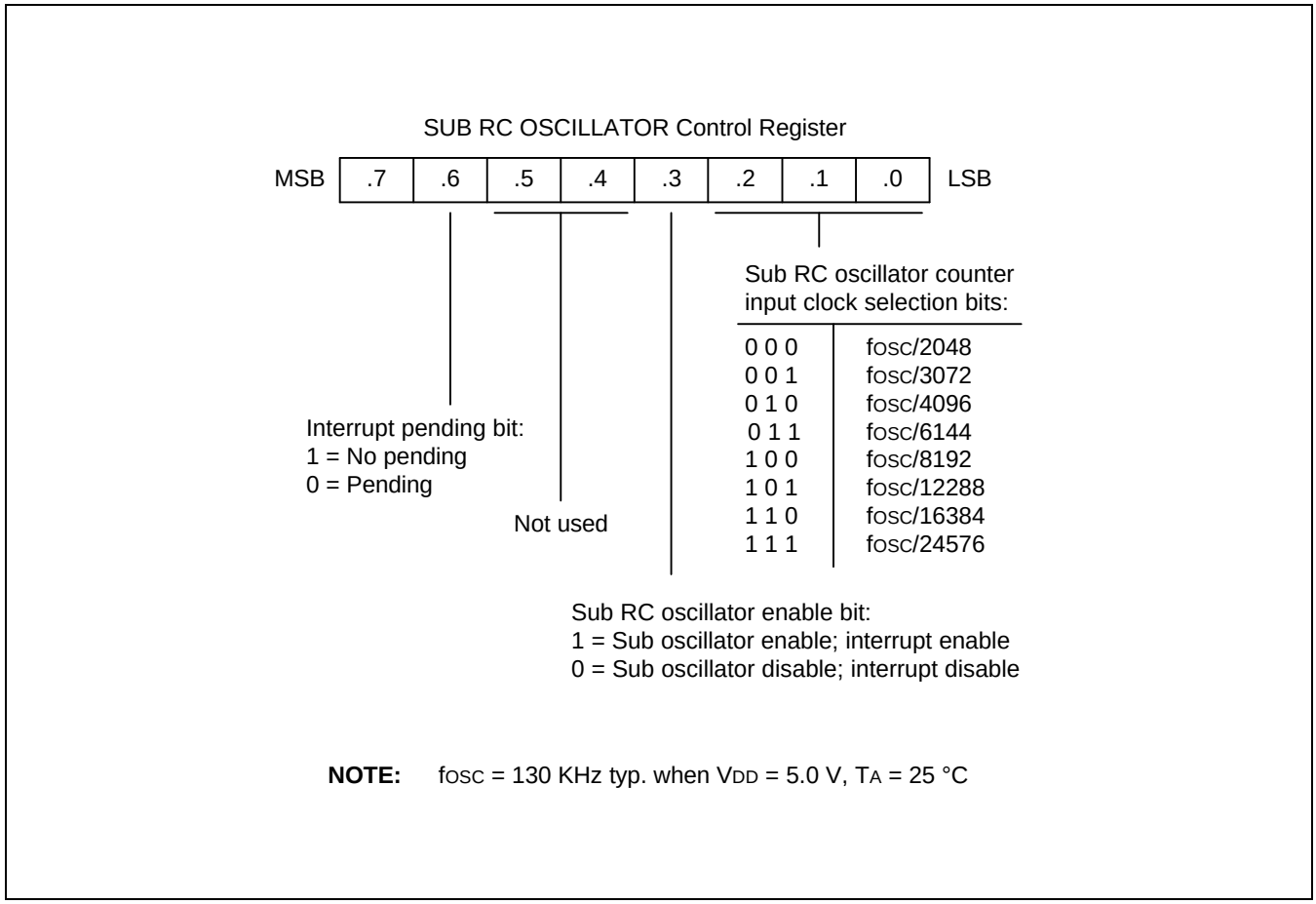

**Figure 12-3. Comparator Functional Block Diagram**

# 13 SUB RC OSCILLATOR

## OVERVIEW

The S3C9654/C9658/P9658 have a programmable SUB RC OSCILLATOR. During IDLE or STOP, programmable SUB RC OSCILLATOR generated interrupt using SUB RC OSCILLATOR control register (SUBCON).

### SUB RC OSCILLATOR CONTROL REGISTER (SUBCON)




**NOTE:** $f_{OSC}$ = 130 KHz typ. when $V_{DD}$ = 5.0 V, $T_A$ = 25 °C

**Figure 13-1. SUB RC OSCILLATOR Control Register**

# NOTES

# 14 LVR (LOW VOLTAGE RESET)

## OVERVIEW

The S3C9654/C9658/P9658 have a LVR (Low Voltage Reset) for power on reset and voltage reset.

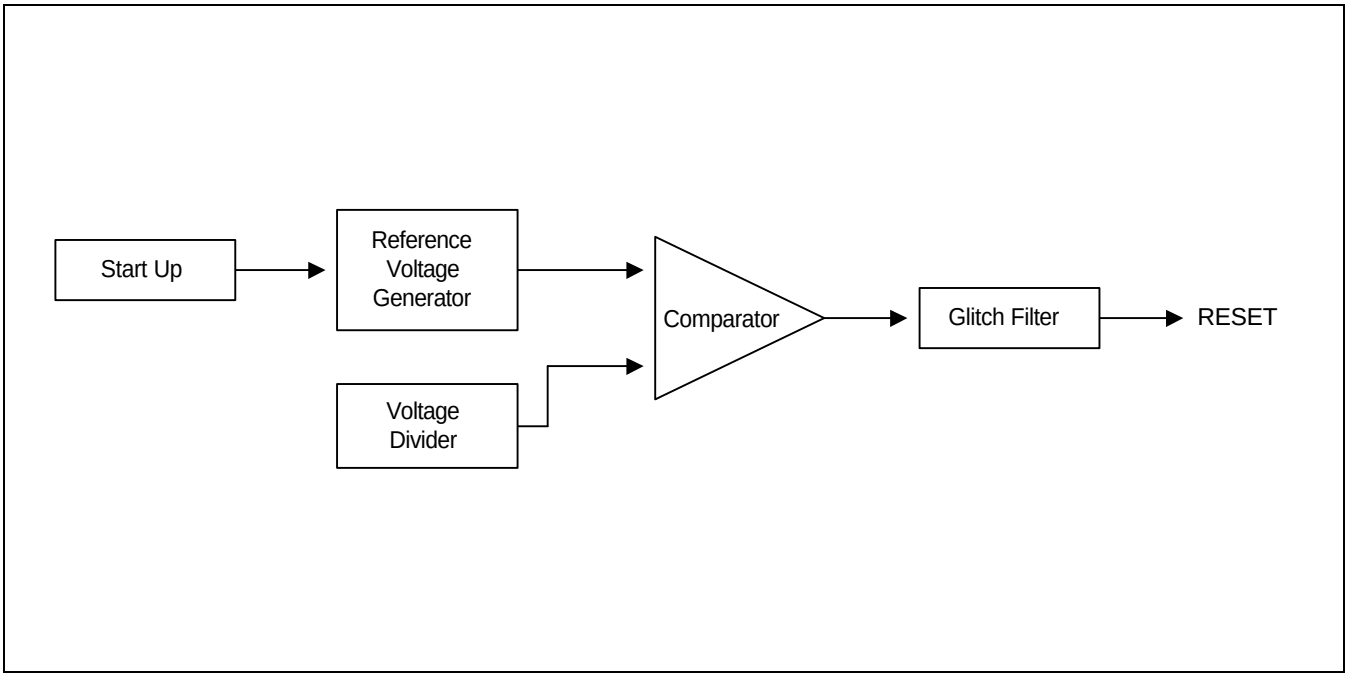
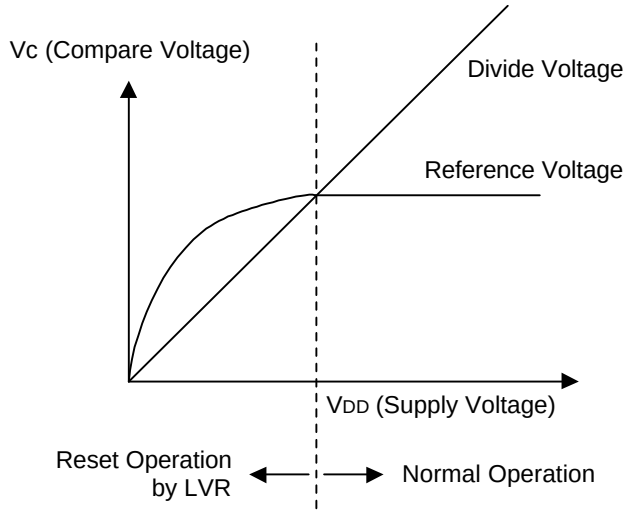



**Figure 14-1. LVR Architecture**

— Low Voltage Reset generated RESET signal.
— Start Up Circuit: Start up reference voltage generator circuit when device powered.
— Reference Voltage Generator: Supply Voltage indeoendent reference voltage generator.
— Voltage Divider: Divide supply voltage by “N”
— Comparator: Compare reference voltage and divided voltage.
— Glitch Filter: Remove glitch and noise signal.

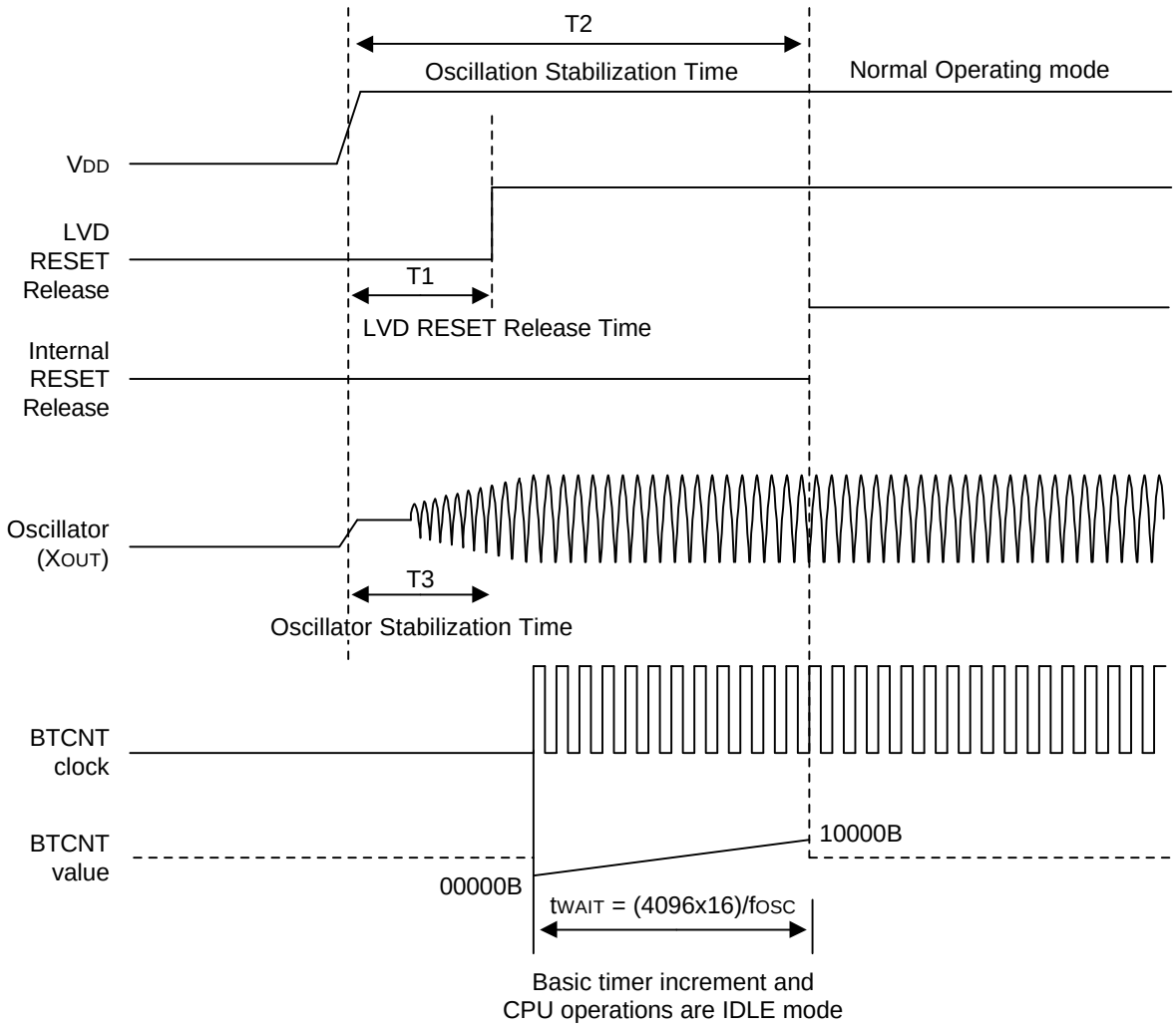



**NOTES:**

1. LVR Operation Voltage Range: 2.3 V-6.0 V
2. LVR Detection Voltage Range: 3.4 V ±0.4 V
3. LVR Current Consumption:
   Less then 10 uA (normally 5 uA)
4. LVR Powered Reset Release Time:
   more then 500 usec (LVR only, typical)
5. LVR Simulation Conditions (Hspice Simulation)
   Temp: 0 °C - 80 °C
   Process Veriation: Worst to best conditions
   Test Voltage: 0.0 V - 7.0 V
   Powered Slew Rate: 5 V/1 usec- 5 V/100 msec

**Figure 14-2. LVR Characteristics**

## LVR AND POWER ON RESET OPERATIONS




**NOTES:**
1. T1 = 500 usc (at normal)
2. T2 = T1 + (4096 x 16)/fOSC

**Figure 14-3. LVR and Power On RESET Operation**

# NOTES

# 15 ELECTRICAL DATA

## OVERVIEW

In this section, the following S3C9654/C9658/P9658 electrical characteristics are presented in tables and graphs:

— Absolute maximum ratings
— D.C. electrical characteristics
— I/O capacitance
— A.C. electrical characteristics
— Oscillator characteristics
— Operating voltage range
— Oscillation stabilization time
— Clock timing measurement points at $X_{IN}$
— Data retention supply voltage in Stop mode
— Stop mode release timing when initiated by a RESET
— Stop mode release timing when initiated by an external interrupt
— Characteristic curves
— Comparator Electrical Characteristics

**Table 15-1. Absolute Maximum Ratings**

($T_A$ = 25°C)

| Parameter | Symbol | Conditions | Rating | Unit |
|---|---|---|---|---|
| Supply voltage | $V_{DD}$ | – | – 0.3 to + 6.5 | V |
| Input voltage | $V_I$ | All ports | – 0.3 to $V_{DD}$ + 0.3 | V |
| Output voltage | $V_O$ | All output ports | – 0.3 to $V_{DD}$ + 0.3 | V |
| Output current high | $I_{OH}$ | One I/O pin active | – 18 | mA |
| | | All I/O pins active | – 60 | |
| Output current low | $I_{OL}$ | One I/O pin active (except P0.0) | + 30 | mA |
| | | Total pin current for ports 0, 1, 2 (except P0.0) | + 100 | |
| | | P0.0 | + 50 | |
| Operating temperature | $T_A$ | – | 0 to + 85 | °C |
| Storage temperature | $T_{STG}$ | – | – 60 to + 150 | |

## Table 15-2. D.C. Electrical Characteristics

($T_A$ = 0°C to + 85°C, $V_{DD}$ = 4.0 V to 5.25 V)

| Parameter | Symbol | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| Input high voltage | $V_{IH1}$ | All input pins except $V_{IH2}$, D+, D– | 0.8 $V_{DD}$ | – | $V_{DD}$ | V |
| | $V_{IH2}$ | $X_{IN}$ | $V_{DD}$ – 0.5 | | $V_{DD}$ | |
| Input low voltage | $V_{IL1}$ | All input pins except $V_{IL2}$, D+, D– | – | – | 0.2 $V_{DD}$ | |
| | $V_{IL2}$ | $X_{IN}$ | – | – | 0.4 | |
| Output high voltage | $V_{OH}$ | $V_{DD}$ = 4.0 V–5.25 V<br>$I_{OH}$ = – 200 µA<br>All output ports except D+, D– | $V_{DD}$ – 1.0 | – | – | |
| Output low voltage | $V_{OL}$ | $V_{DD}$ = 4.0 V–5.25 V<br>$I_{OL}$ = 2 mA<br>All output ports except D+, D–, P0.0 | – | – | 0.4 | |
| Output low Current | $I_{OL}$ | $V_{OL}$ = 0.4 V | | 50(4) | | mA |
| Input high leakage current | $I_{LIH1}$ | $V_{IN}$ = $V_{DD}$<br>All inputs except $I_{LIH2}$<br>except D+, D–, $X_{OUT}$ | – | – | 3 | µA |
| | $I_{LIH2}$ | $V_{IN}$ = $V_{DD}$, $X_{IN}$ | – | – | 20 | |
| Input low leakage current | $I_{LIL1}$ | $V_{IN}$ = 0 V<br>All inputs except $I_{LIL2}$<br>except D+, D–, $X_{OUT}$ | – | – | – 3 | |
| | $I_{LIL2}$ | $V_{IN}$ = 0 V, $X_{IN}$ | – | – | – 20 | |
| Output high leakage current | $I_{LOH}$ | $V_{OUT}$ = $V_{DD}$<br>All output pins except D+, D– | – | – | 3 | |
| Output low leakage current | $I_{LOL}$ | $V_{OUT}$ = 0 V<br>All output pins except D+, D–<br>$X_{OUT}$, P0.0 | – | – | – 3 | |
| Pull-up resistors | $R_{L1}$ | $V_{IN}$ = 0 V, $V_{DD}$ = 5.0 V,<br>Port 0, Port 1 | 25 | 50 | 100 | KΩ |
| | $R_{L2}$ | $V_{IN}$ = 0 V, $V_{DD}$ = 5.0 V,<br>Port 2 | – | 4.3 | – | |
| Supply current | $I_{DD1}$ | Normal operation mode,<br>$V_{DD}$ = 4.0 V – 5.25 V<br>6 MHz, CPU clock | – | 6.5 | 15 | mA |
| | $I_{DD2}$ | IDLE mode<br>$V_{DD}$ = 4.0 V – 5.25 V<br>6 MHz, CPU clock | – | 2 | 4 | |
| | $I_{DD3}$ | Stop mode, oscillator stop<br>$V_{DD}$ = 5 V ± 10% | – | 20 | 50 | µA |

**NOTES:**
1. Supply current does not include current drawn through internal pull-up resistors or external output current load.
2. This parameter is guaranteed, but not tested (include D+, D–).
3. Only in 4.0 V to 5.25 V, D+ and D– satisfy the USB spec 1.0.
4. P0.0 designed for direct LED current sink, see the SNKCON resistor and Figure 1-9 (Page 1-9).

**Table 15-3. Input/Output Capacitance**

($T_A$ = 0°C to + 85°C, $V_{DD}$ = 0 V)

| Parameter | Symbol | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| Input capacitance | $C_{IN}$ | f = 1 MHz; unmeasured pins are connected to $V_{SS}$ Except $X_{IN}$, $X_{OUT}$ | – | – | 10 | pF |
| Output capacitance | $C_{OUT}$ | | | | | |
| I/O capacitance | $C_{IO}$ | | | | | |
| XI/XO capacitance | $C_{XI}$, $C_{XO}$ | $X_{IN}$, $X_{OUT}$ | – | 33 | – | |

**Table 15-4. A.C. Electrical Characteristics**

($T_A$ = 0°C to + 85°C, $V_{DD}$ = 4.0 V to 5.25 V)

| Parameter | Symbol | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| Noise filter | $t_{NF1H}$, $t_{NF1L}$ | P1 (RC delay) | 100 | – | 200 | ns |

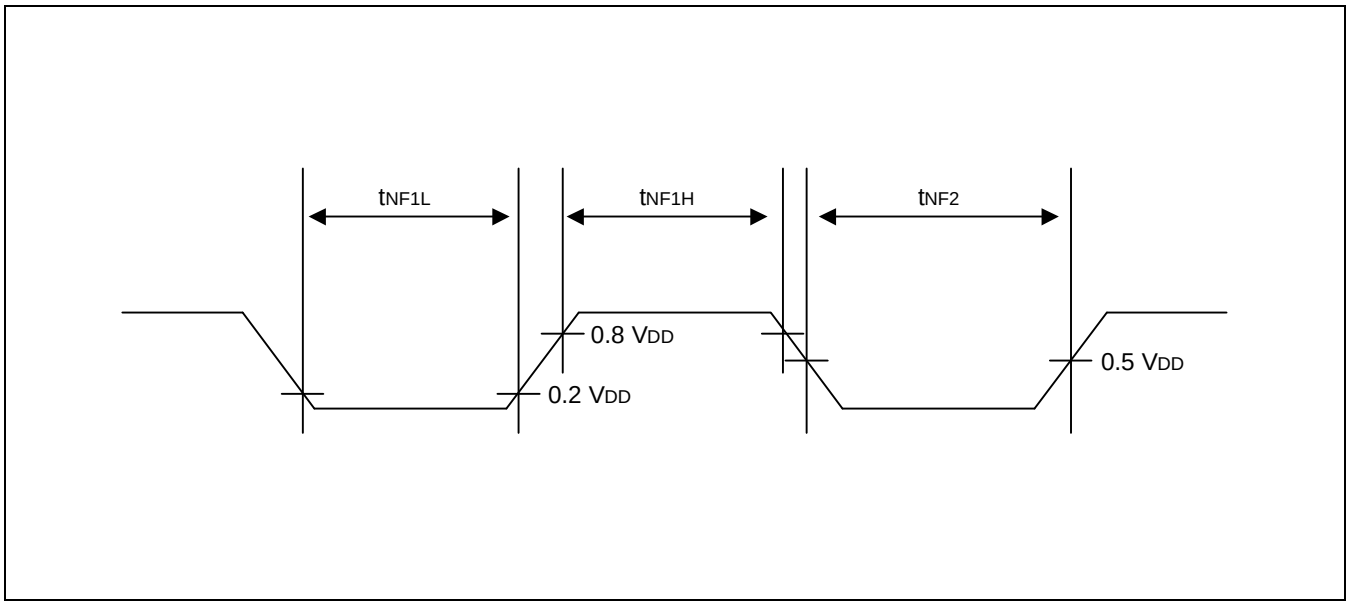



**Figure 15-1. Nose Filter Timing Measurement Points**

**Table 15-5. Oscillator Characteristics**

($T_A$ = 0°C + 85°C)

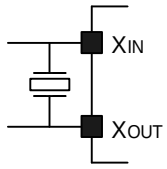
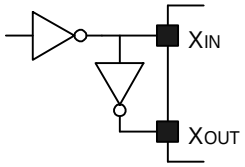
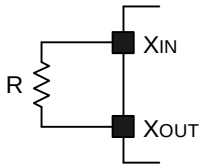
| Oscillator | Clock Circuit | Test Condition | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| Main crystal Main ceramic ($f_{OSC}$) |  | Oscillation frequency<br>$V_{DD}$ = 4.0 V–5.25 V | – | 6.0 | – | MHz |
| External clock |  | Oscillation frequency<br>$V_{DD}$ = 4.0 V–5.25 V | – | 6.0 | – | MHz |
| RC oscillator |  | Oscillation frequency<br>$V_{DD}$ = 5.0 V<br>R = 22.6 K<br>R = 8.8 K<br>R = 3.2 K | <br><br>–<br>–<br>– | <br><br>1.0<br>2.0<br>4.0 | <br><br>–<br>–<br>– | MHz |

**Table 15-6. Oscillation Stabilization Time**

($T_A$ = 0°C + 85°C, $V_{DD}$ = 4.0 V to 5.25 V)

| Oscillator | Test Condition | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| Main crystal<br>Main ceramic | $V_{DD}$ = 4.0 V to 5.25 V, $f_{OSC}$ > 6.0 MHz<br>(Oscillation stabilization occurs when $V_{DD}$ is equal to the minimum oscillator voltage range.) | – | – | 10 | ms |
| Oscillator stabilization wait time | $t_{WAIT}$ stop mode release time by a reset | – | $2^{16}/f_{OSC}$ | – | |
| | $t_{WAIT}$ stop mode release time by an interrupt | – | – | – | |

**NOTE:** The oscillator stabilization wait time, $t_{WAIT}$, when it is released by an interrupt, is determined by the setting in the basic timer control register, BTCON.

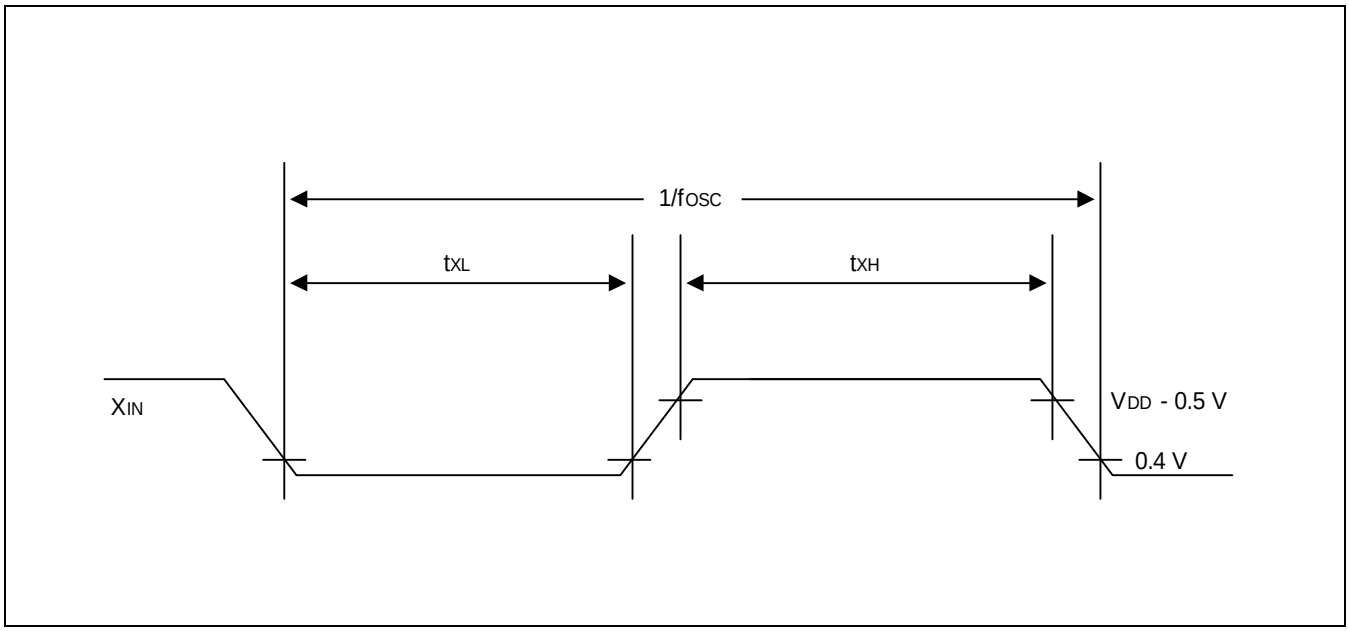



**Figure 15-2. Clock Timing Measurement Points at $X_{IN}$**

**Table 15-7. Data Retention Supply Voltage in Stop Mode**

($T_A$ = 0°C to + 70°C)

| Parameter | Symbol | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| Data retention supply voltage | $V_{DDDR}$ | Stop mode | 2.0 | – | 6 | V |
| Data retention supply current | $I_{DDDR}$ | Stop mode; $V_{DDDR}$ = 2.0 V | – | – | 5 | µA |

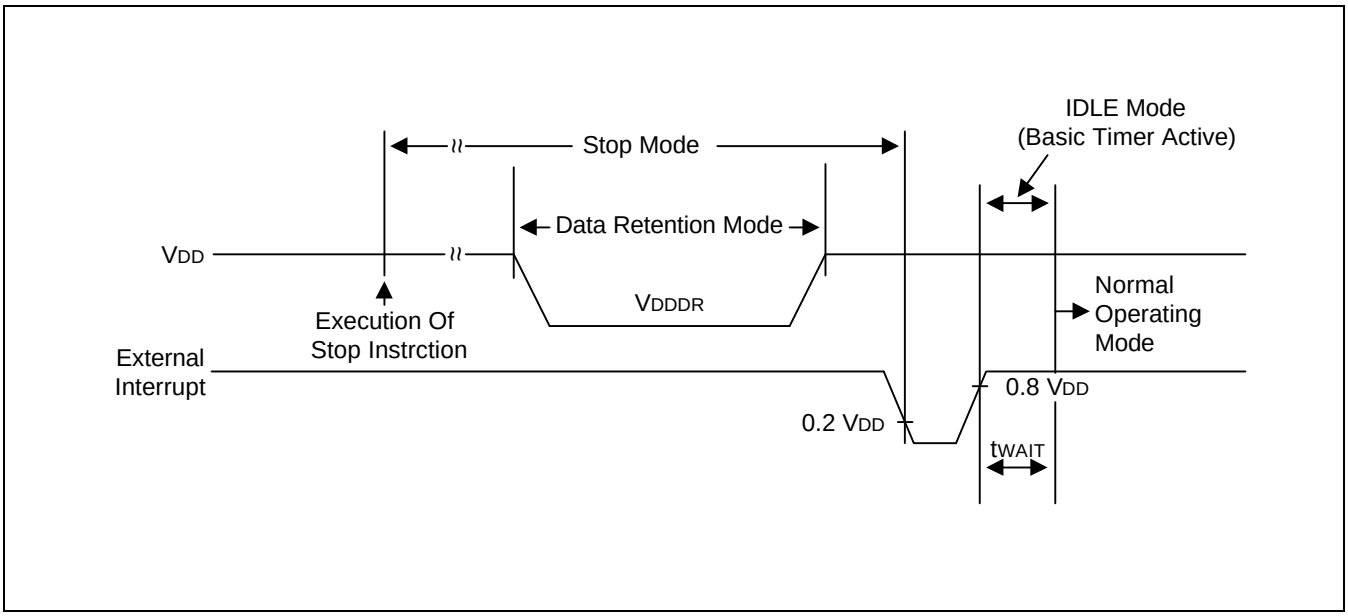



**Figure 15-3. Stop Mode Release Timing When Initiated by an External Interrupt**

**Table 15-8. Comparator Electrical Characteristics**

($T_A$ = 0°C to + 85°C, $V_{DD}$ = 4.0 V to 5.25 V)

| Parameter | Symbol | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| Conversion time (1) | tCON | – | – | 6 × 12 or 6 × 192 | – | $f_{CPU}$ |
| Comparator input voltage | $V_{ICN}$ | – | $V_{SS}$ | – | $V_{DD}$ | V |
| Comparator input impedance | $R_{CN}$ | – | 2 | 1000 | – | MΩ |
| Comparator reference voltage | $V_{REF}$ | – | 1.8 | – | $V_{DD}$ | V |
| Comparator input current | $I_{CIN}$ | $V_{DD}$ = 5 V | – 3 | – | 3 | µA |
| Reference input current | $I_{REF}$ | $V_{DD}$ = 5 V | – 3 | – | 3 | µA |
| Comparator block current (2) | $I_{COM}$ | $V_{DD}$ = 5.5 V | – | 1 | 2 | mA |
| | | $V_{DD}$ = 4.5 V | | 0.5 | 1 | mA |
| | | $V_{DD}$ = 5 V (when power down mode) | | 100 | 500 | nA |

**NOTES:**
1. Conversion time is the time required from the moment a conversion operation starts until it ends.
2. $I_{COM}$ is an operating current during conversion.

**Table 15-9. Low Speed Source Electrical Characteristics (USB)**

($T_A$ = 0°C to + 85°C, Internal Voltage Regulator Output $V_{33OUT}$ = 2.8 V to 3.6 V, typ 3.3 V)

| Parameter | Symbol | Conditions | Min | Max | Unit |
|---|---|---|---|---|---|
| Transition Time: | | | | | |
| Rise Time | Tr | CL = 200 pF | 75 | – | ns |
| | | CL = 650 pF | – | 300 | |
| Fall Time | Tf | CL = 200 pF | 75 | – | |
| | | CL = 650 pF | – | 300 | |
| Rise/Fall Time Matching | Trfm | (Tr/Tf) CL = 50 pF | 80 | 125 | % |
| Output Signal Crossover Voltage | Vcrs | CL = 50 pF | 1.3 | 2.0 | V |
| Internal Voltage Regulator Output Voltage | $V_{33OUT}$ | $V_{DD}$ = 4.0 – 5.25 V | 2.8 | 3.6 | V |

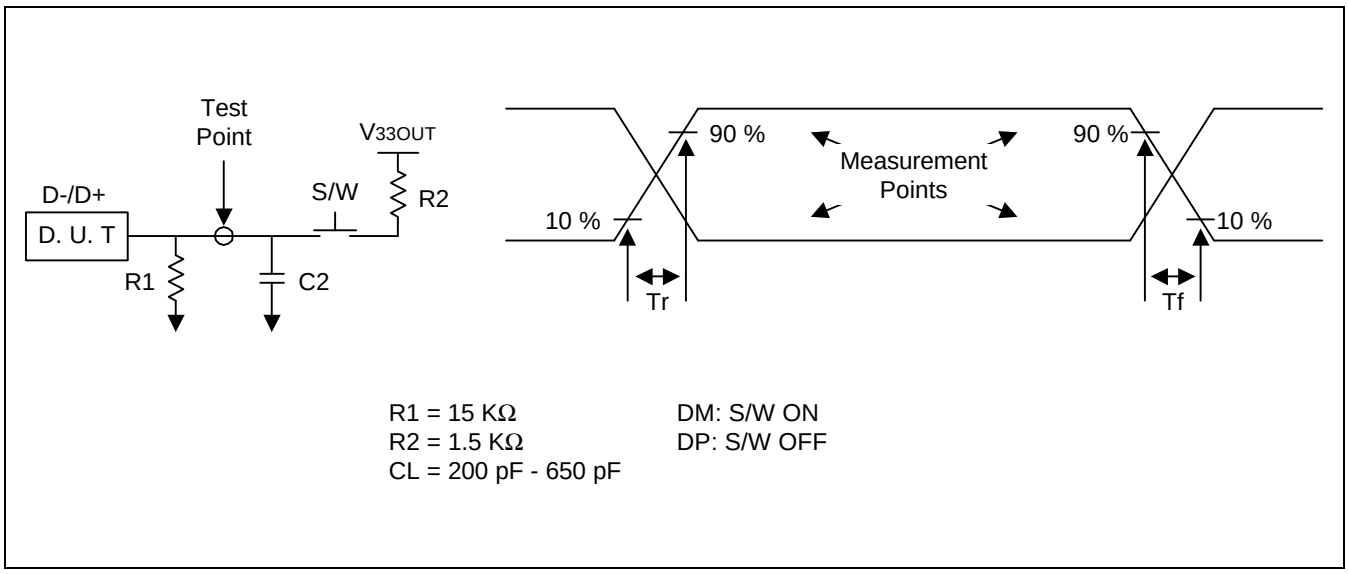



**Figure 15-4. USB Data Signal Rise and Fall Time**

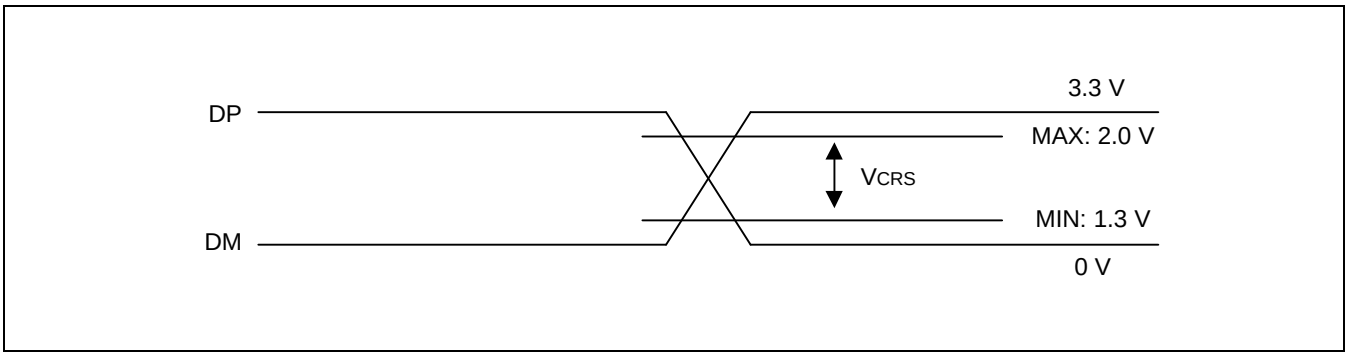


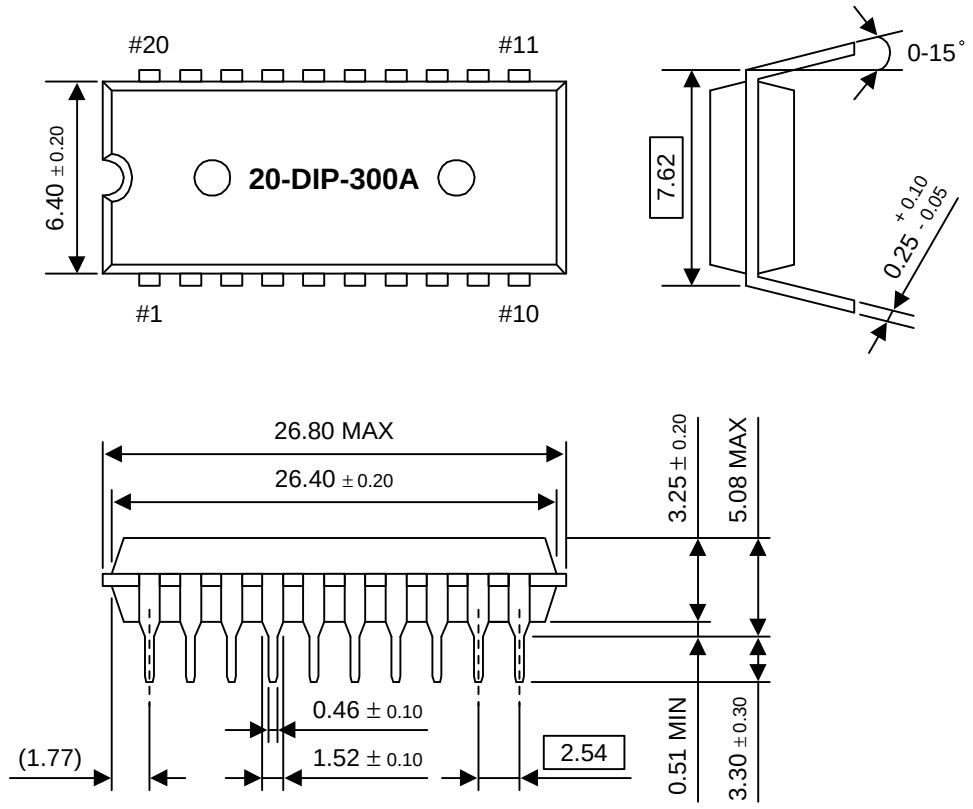

**Figure 15-5. USB Output Signal Crossover Point Voltage**

# 16 MECHANICAL DATA

## OVERVIEW

This section contains the following information about the device package:

— Package dimensions in millimeters
— Pad diagram




**NOTE:** Dimensions are in millimeters.

**Figure 16-1. 20-DIP 300A Package Dimensions**

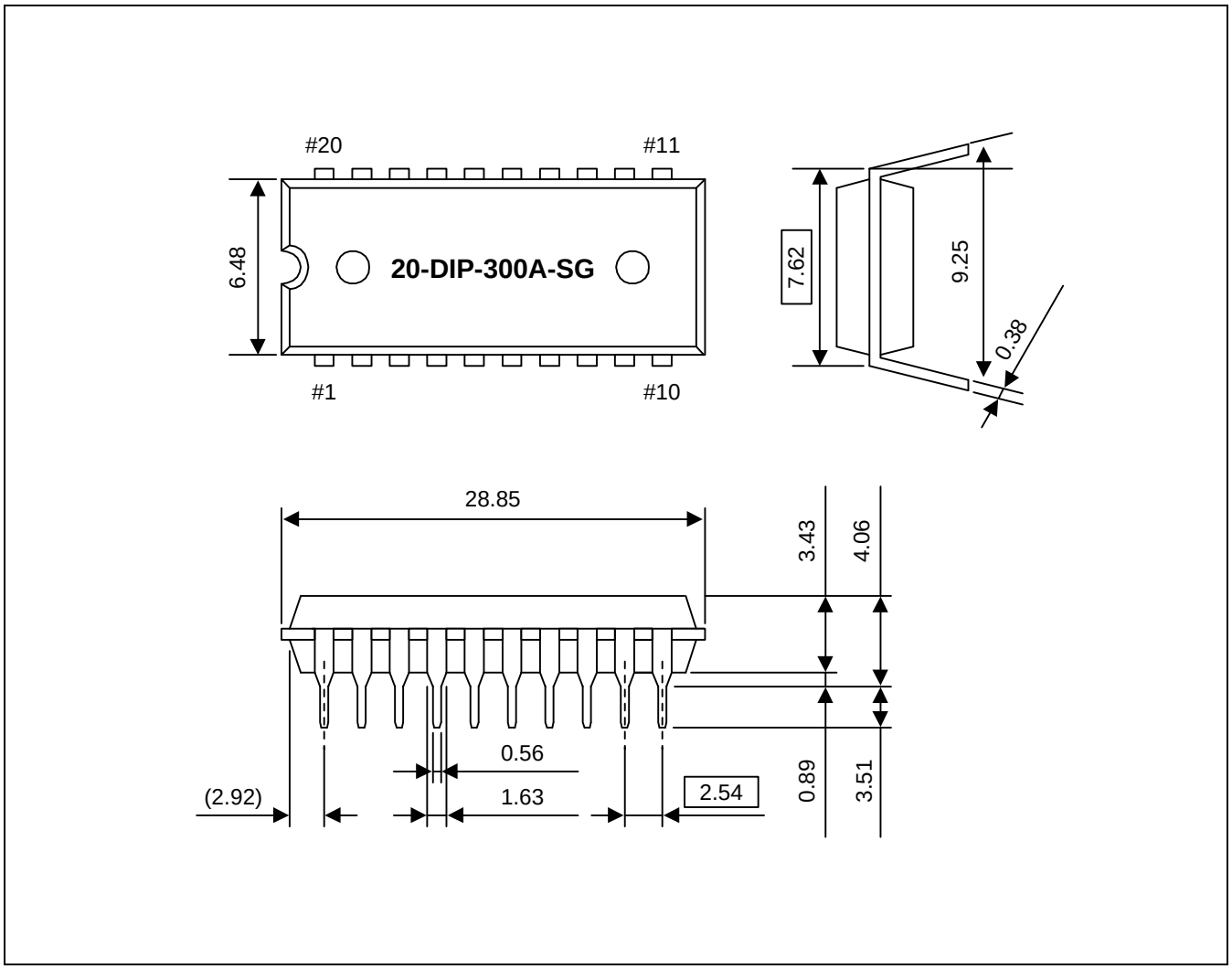
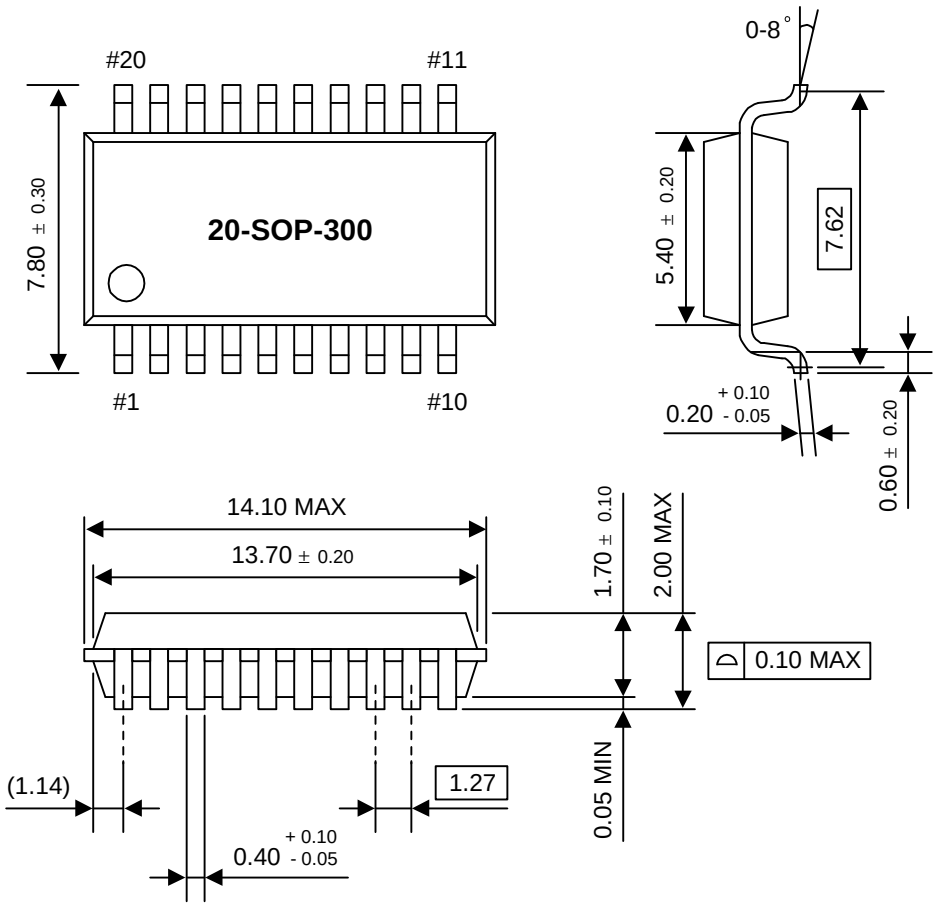



**Figure 16-2. 20-DIP-300A-SG Package Dimensions**




**NOTE:** Dimensions are in millimeters.

**Figure 16-3. 20-SOP-300 Package Dimensions**

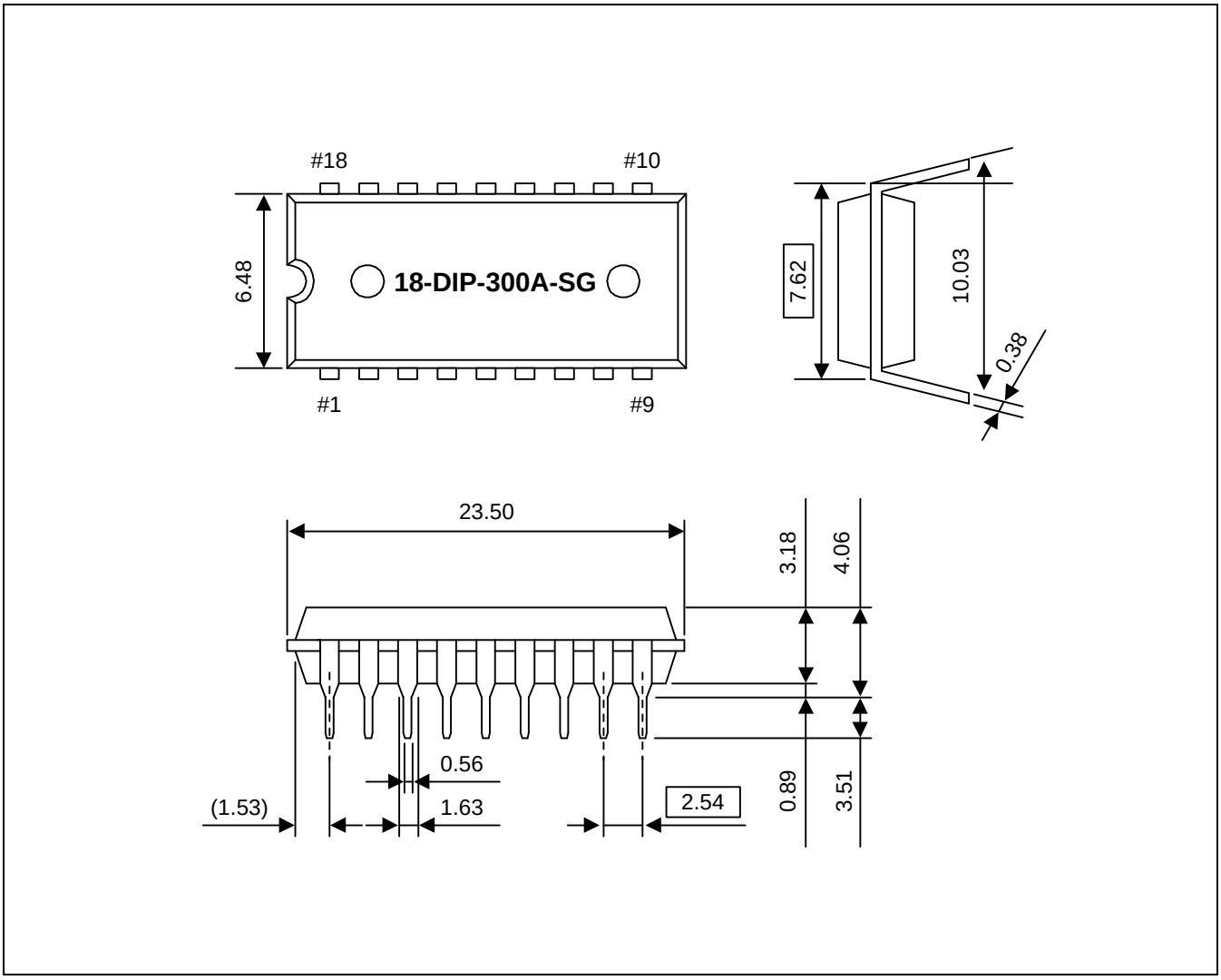



**Figure 16-4. 18-DIP-300A-SG Package Dimensions**

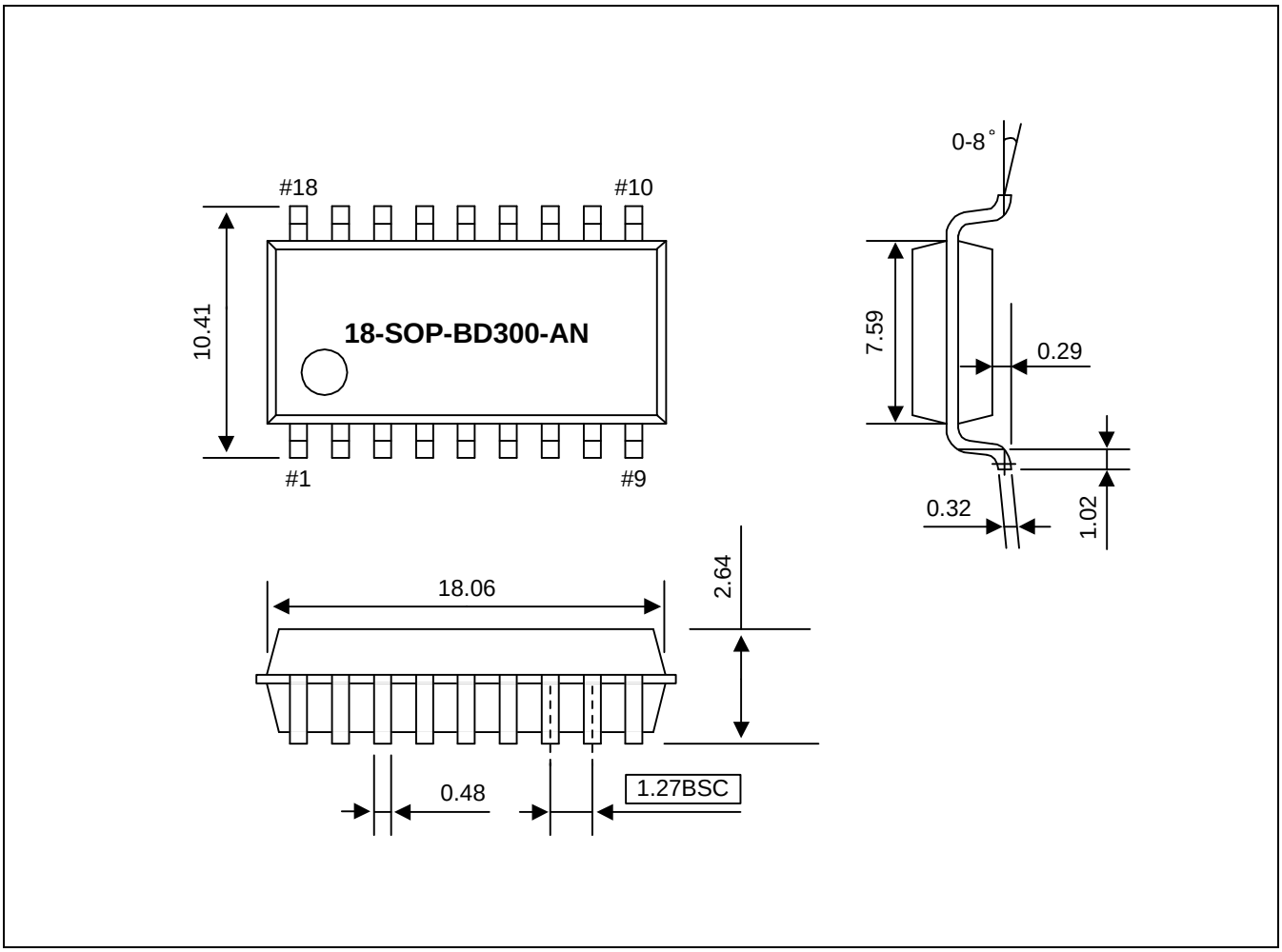



**Figure 16-5. 18-SOP-BD300-AN Package Dimensions**

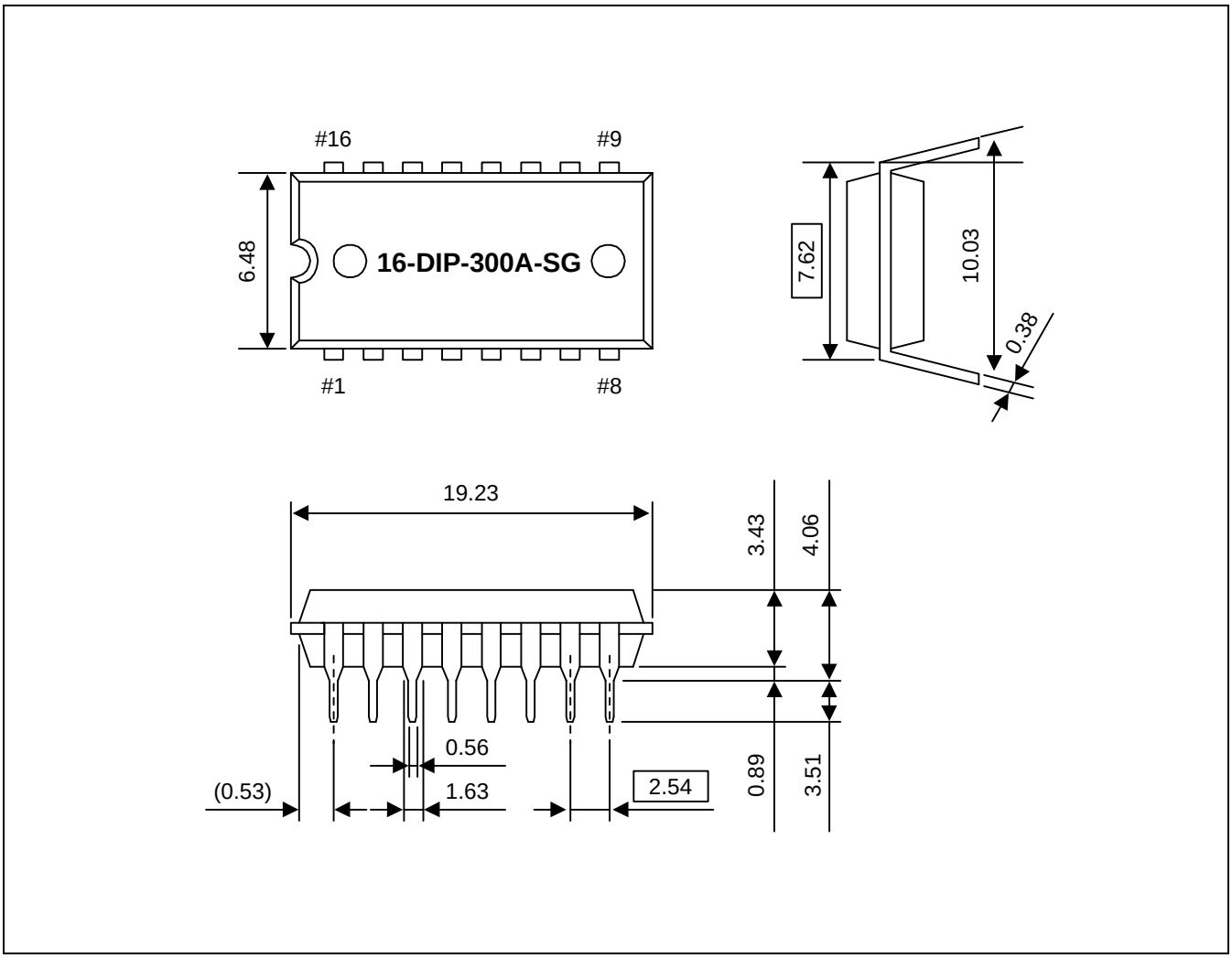



**Figure 16-6. 16-DIP-300A-SG Package Dimensions**

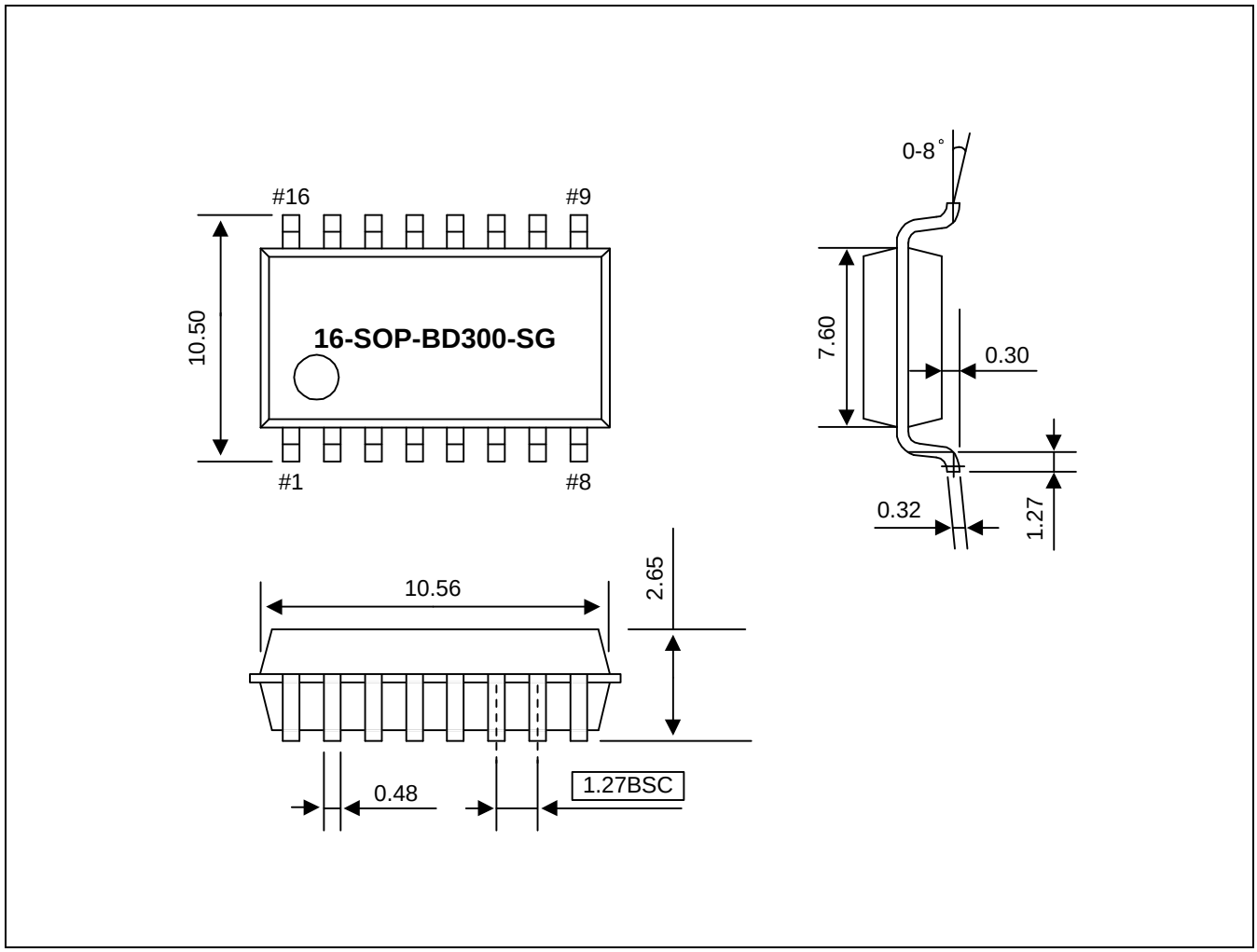



**Figure 16-7. 16-SOP-BD300-SG Package Dimensions**

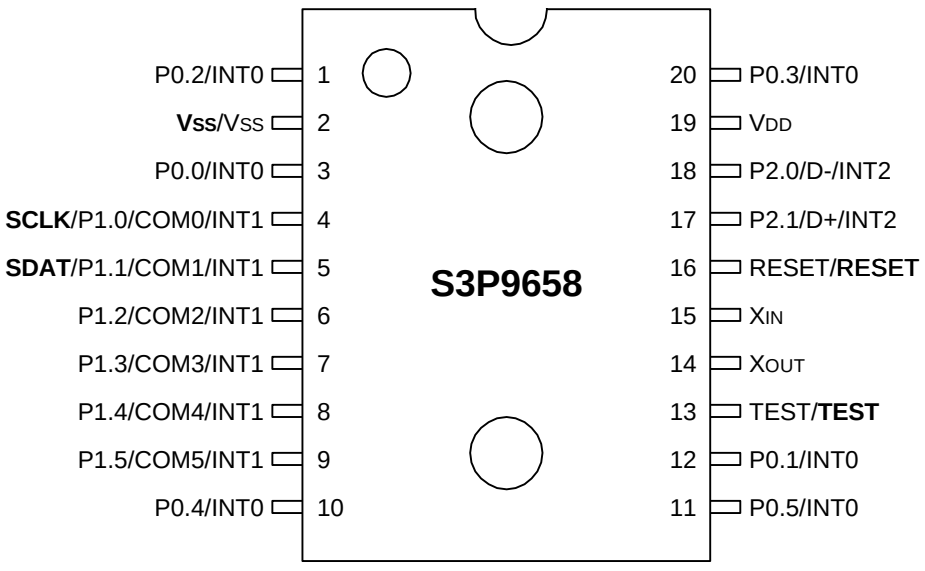
# NOTES

# 17 S3P9658 OTP

## OVERVIEW

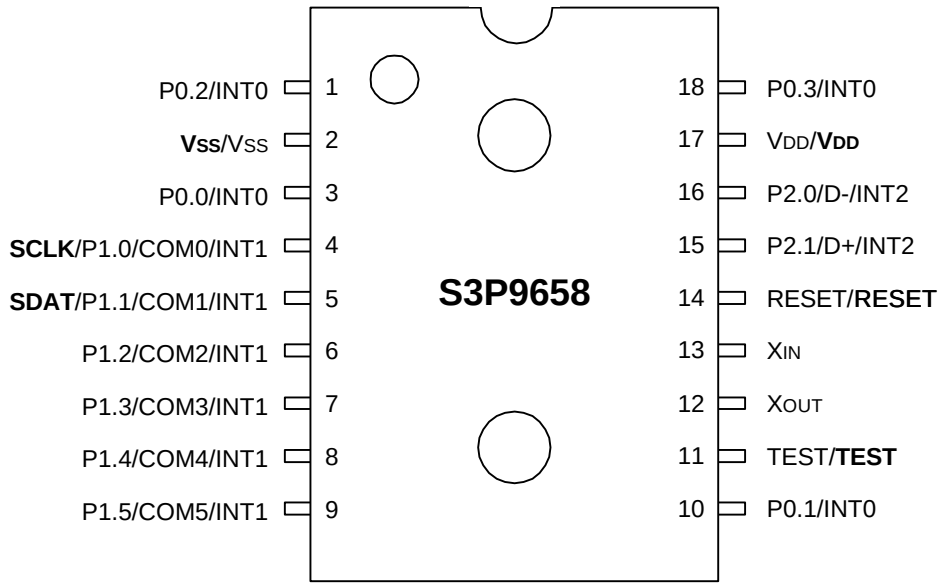
The S3P9658 single-chip CMOS microcontroller is the OTP (One Time Programmable) version of the S3P9658 microcontroller. It has an on-chip OTP ROM instead of masked ROM. The EPROM is accessed by serial data format.

The S3P9658 is fully compatible with the S3P9658, both in function and in pin configuration. Because of its simple programming requirements, the S3P9658 is ideal for use as an evaluation chip for the S3P9658.



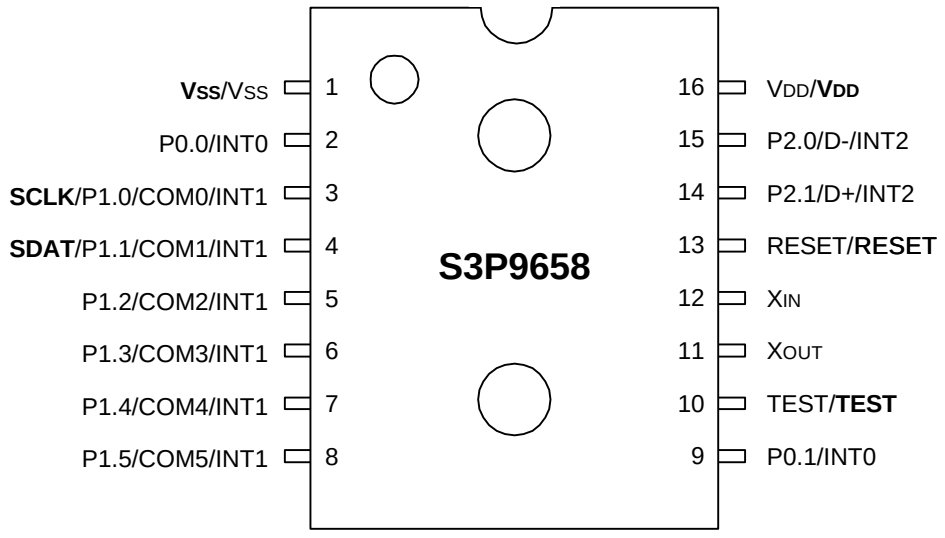

**NOTE:** The bold is indicate an OTP pin name.

**Figure 17-1. S3P9658 Pin Assignments (20 Pin)**




**NOTE:** The bold is indicate an OTP pin name.

**Figure 17-2. S3P9658 Pin Assignments (18 Pin)**




**NOTE:** The bold is indicate an OTP pin name.

**Figure 17-3. S3P9658 Pin Assignments (16 Pin)**

**Table 17-1. Descriptions of Pins Used to Read/Write the EPROM**

| Main Chip | During Programming | | | |
|---|---|---|---|---|
| Pin Name | Pin Name | Pin Number (20 DIP) | I/O | Function |
| P1.0 | SDAT | 5 | I/O | Serial Data Pin (Output when reading, Input when writing) Input and Push-pull Output Port can be assigned |
| P1.1 | SCLK | 4 | I/O | Serial Clock Pin (Input Only Pin) |
| RESET | RESET | 16 | I | 0 V : OTP write and test mode<br>5 V : Operating mode |
| TEST | $V_{PP}$ (TEST) | 13 | I | Chip Initialization and EPROM Cell Writing Power Supply Pin (Indicates OTP Mode Entering) When writing 12.5 V is applied and when reading. |
| $V_{DD}/V_{SS}$ | $V_{DD}/V_{SS}$ | 19/2 | I | Logic Power Supply Pin. |

**Table 17-2. Comparison of S3P9658 and S3C9654/C9658 Features**

| Characteristic | S3P9658 | S3C9654/C9658 |
|---|---|---|
| Program Memory | 8 K-byte EPROM | 4/8 K-byte mask ROM |
| Operating Voltage ($V_{DD}$) | 4.0 V to 5.25 V | 4.0 V to 5.25 V |
| OTP Programming Mode | $V_{DD}$ = 5 V, $V_{PP}$ (TEST) = 12.5 V | |
| Pin Configuration | 20/18/16 DIP, 20/18/16 SOP | 20/18/16 DIP, 20/18/16 SOP, 16SSOP |
| EPROM Programmability | User Program 1 time | Programmed at the factory |

## OPERATING MODE CHARACTERISTICS

When 12.5 V is supplied to the $V_{PP}$ (RESET) pin of the S3P9658, the EPROM programming mode is entered. The operating mode (read, write, or read protection) is selected according to the input signals to the pins listed in Table 14-3 below.

**Table 17-3. Operating Mode Selection Criteria**

| $V_{DD}$ | $V_{PP}$ (RESET) | REG/MEM | Address (A15-A0) | R/W | Mode |
|---|---|---|---|---|---|
| 5 V | 5 V | 0 | 0000H | 1 | EPROM read |
| | 12.5 V | 0 | 0000H | 0 | EPROM program |
| | 12.5 V | 0 | 0000H | 1 | EPROM verify |
| | 12.5 V | 1 | 0E3FH | 0 | EPROM read protection |

**NOTE:** "0" means Low level; "1" means High level.

# NOTES

# 18 DEVELOPMENT TOOLS

## OVERVIEW

Samsung provides a powerful and easy-to-use development support system in turnkey form. The development support system is configured with a host system, debugging tools, and support software. For the host system, any standard computer that operates with MS-DOS as its operating system can be used. One type of debugging tool including hardware and software is provided: the sophisticated and powerful in-circuit emulator, SMDS2+, for S3C7, S3C9, S3C8 families of microcontrollers. The SMDS2+ is a new and improved version of SMDS2. Samsung also offers support software that includes debugger, assembler, and a program for setting options.

### SHINE

Samsung Host Interface for in-circuit Emulator, SHINE, is a multi-window based debugger for SMDS2+. SHINE provides pull-down and pop-up menus, mouse support, function/hot keys, and context-sensitive hyper-linked help. It has an advanced, multiple-windowed user interface that emphasizes ease of use. Each window can be sized, moved, scrolled, highlighted, added, or removed completely.

### SAMA ASSEMBLER

The Samsung Arrangeable Microcontroller (SAM) Assembler, SAMA, is a universal assembler, and generates object code in standard hexadecimal format. Assembled program code includes the object code that is used for ROM data and required SMDS program control data. To assemble programs, SAMA requires a source file and an auxiliary definition (DEF) file with device specific information.

### SASM86

The SASM86 is an relocatable assembler for Samsung's S3C9-series microcontrollers. The SASM86 takes a source file containing assembly language statements and translates into a corresponding source code, object code and comments. The SASM86 supports macros and conditional assembly. It runs on the MS-DOS operating system. It produces the relocatable object code only, so the user should link object file. Object files can be linked with other object files and loaded into memory.

### HEX2ROM

HEX2ROM file generates ROM code from HEX file which has been produced by assembler. ROM code must be needed to fabricate a microcontroller which has a mask ROM. When generating the ROM code(.OBJ file) by HEX2ROM, the value 'FF' is filled into the unused ROM area upto the maximum ROM size of the target device automatically.

## TARGET BOARDS

Target boards are available for all S3C9-series microcontrollers. All required target system cables and adapters are included with the device-specific target board.

### OTPs

One times programmable microcontrollers (OTPs) are under development for S3C9654/C9658/P9658 microcontroller.

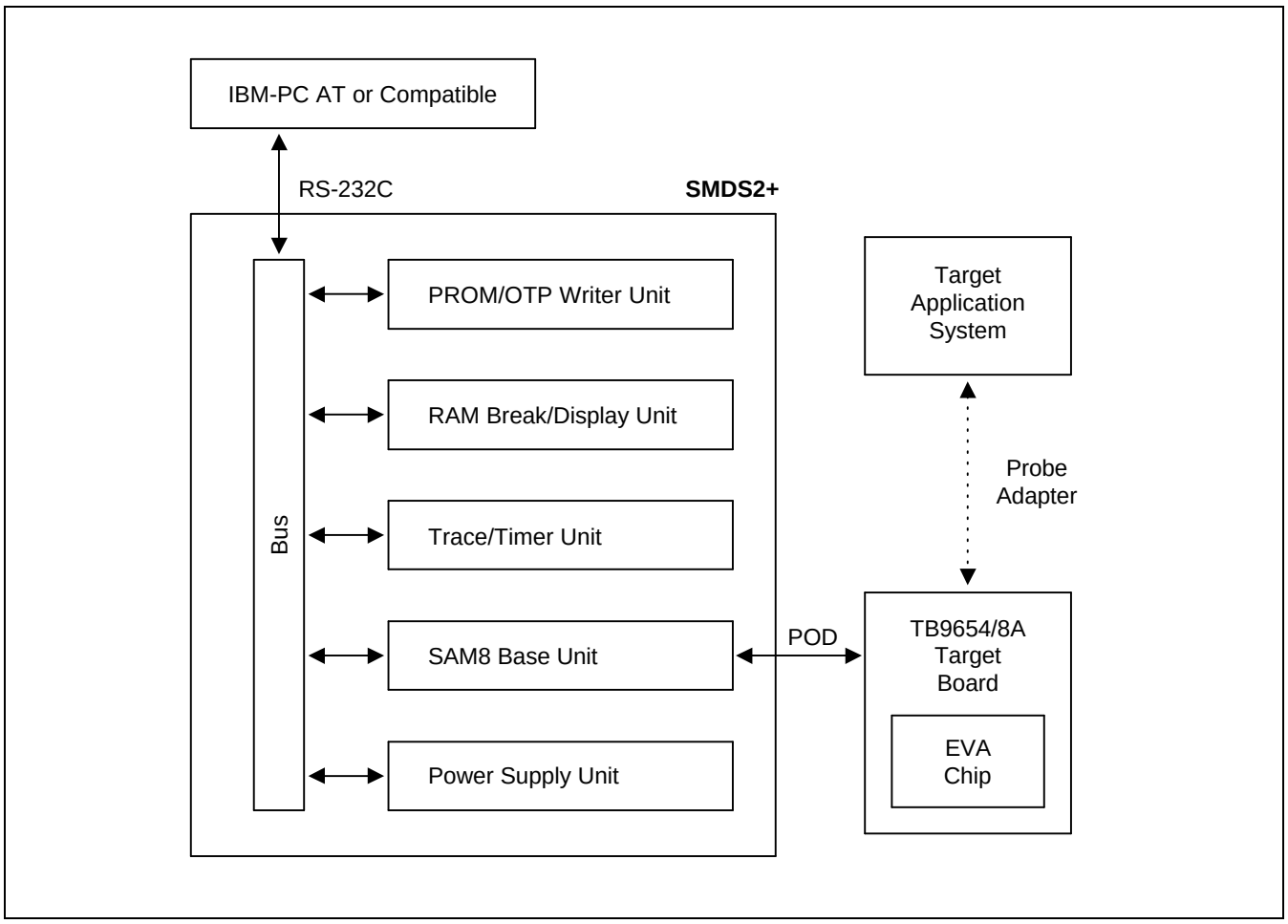



**Figure 18-1. SMDS Product Configuration (SMDS2+)**

## TB9654/8A TARGET BOARD

The TB9654/8A target board is used for the S3C9654/C9658/P9658 microcontrollers. It is supported by the SMDS2+ development system.

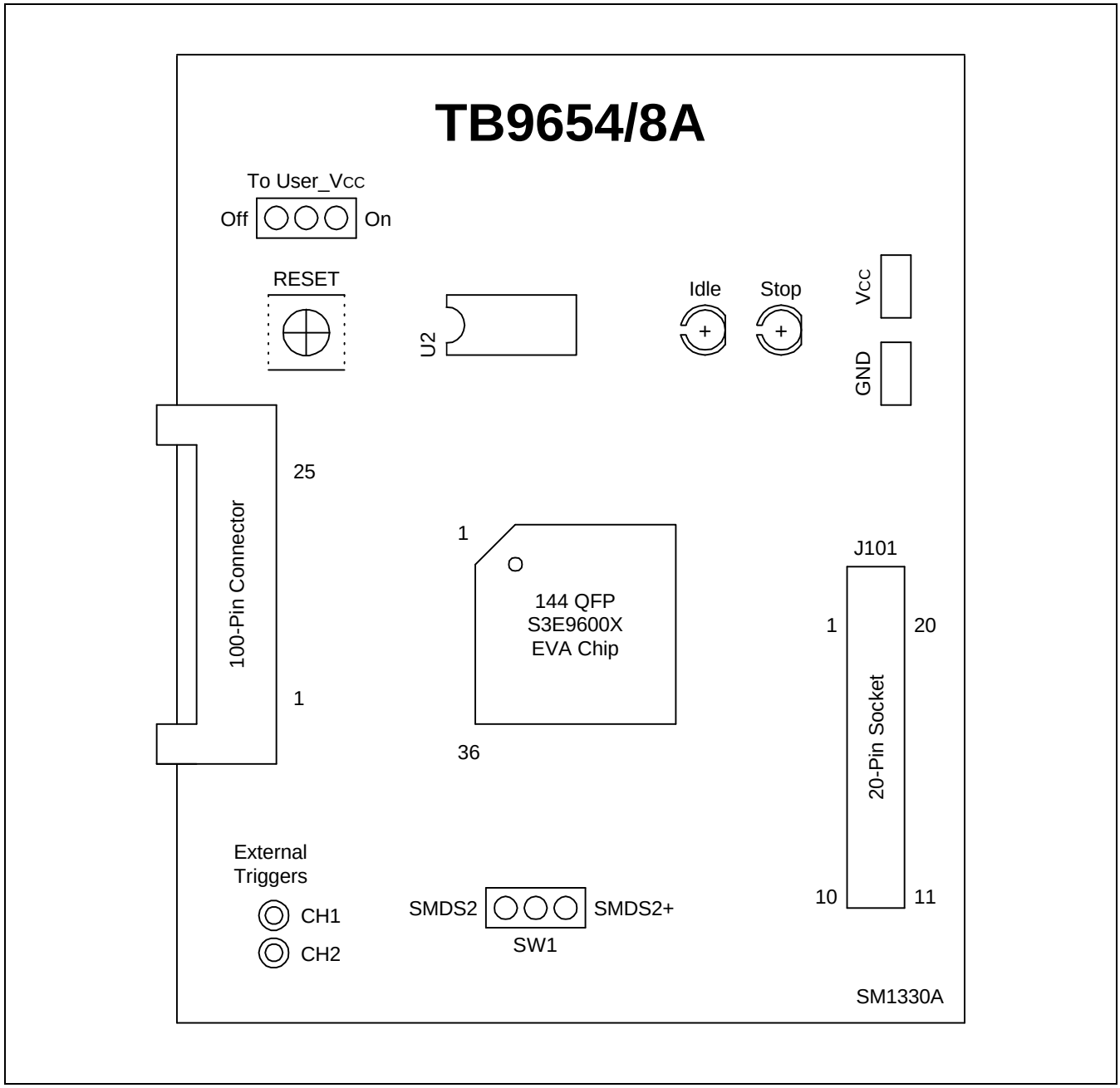
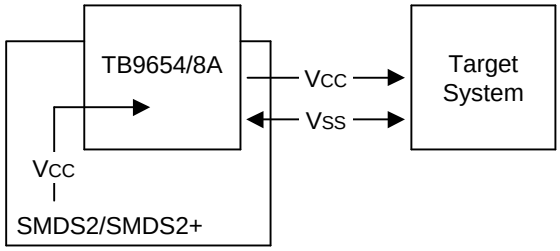
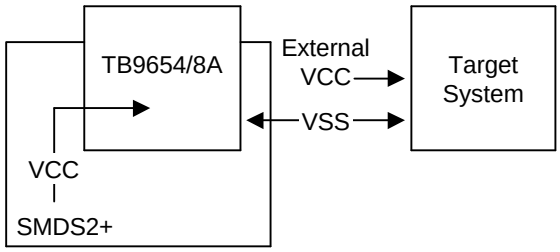



**Figure 18-2. TB9654/8A Target Board Configuration**

**Table 18-1. Power Selection Settings for TB9654/8A**

| 'To User_Vcc' Settings | Operating Mode | Comments |
|---|---|---|
| To User_Vcc<br>Off  On |  | SMDS2/SMDS2+ supplies $V_{CC}$ to the target board (evaluation chip) and the target system. |
| To User_Vcc<br>Off  On |  | SMDS2/SMDS2+ supplies $V_{CC}$ only to the target board (evaluation chip). The target system must have a power supply of its own. |

### SMDS2+ Selection (SAM8)

In order to write data into program memory available in SMDS2+, the target board should be selected for SMDS2+ through a switch as follows. Otherwise, the program memory writing function is not available.

**Table 18-2. The SMDS2+ Tool Selection Setting**

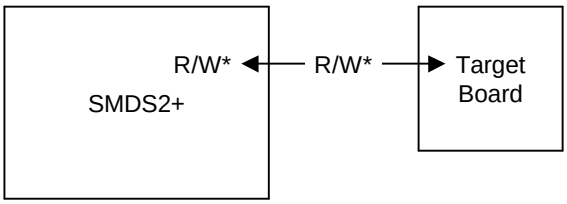
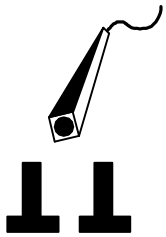
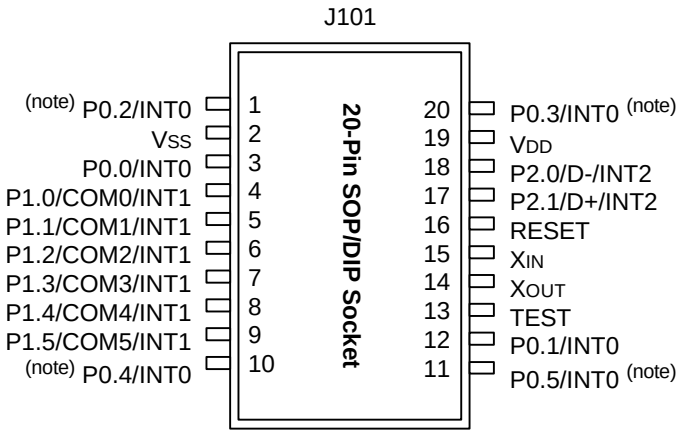
| 'SW1' Setting | Operating Mode |
|---|---|
| SMDS2  SMDS2+ |  |

**Table 18-3. Using Single Header Pins as the Input Path for External Trigger Sources**

| Target Board Part | Comments |
|---|---|
| External Triggers<br>CH1<br>CH2 |  <br>Connector from External Trigger Sources of the Application System<br>You can connect an external trigger source to one of the two external trigger channels (CH1 or CH2) for the SMDS2+ breakpoint and trace functions. |




**NOTE:** 16, 18, and 20 SOP/DIP.

**Figure 18-3. 20-Pin Socket for TB9654/8A**

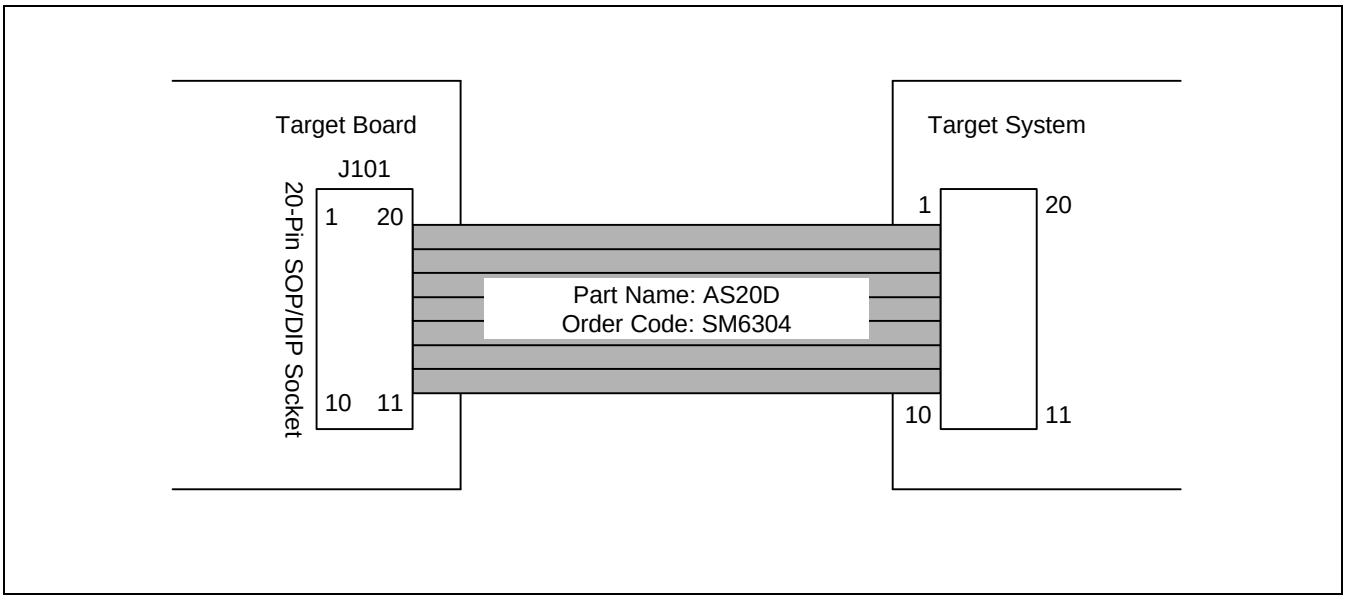



**Figure 18-4. TB9654/8A Adapter Cable for 20-SOP/DIP Package**

# 19 APPENDIX

## SAMSUNG USB CONTROLLER TECHNICAL NOTE

### USB RESET SIGNAL

Application: KS86P/C6104, KS86P/C6504, KS86P/C6408.

Direction: From outside (From host or HUB) to USB controller.

Effect: Reset USB part.
USB reset signal affect to only USB related registers.
USB related registers are changed to reset value as below:

| | |
|---|---|
| USB function address register | → 00h |
| EP0 CSR register | → 00h |
| EP0 byte count register | → 00h |
| EP1 CSR register | → 00h |
| USB interrupt enable register | → 03h |
| USB interrupt pending register | → 00h |
| USB power management register | → 00h |

Notice to MCU: When USB module receive USB reset signal from outside then set USBRST register.
USBRST register → 01h
(This means USB controller have received USB reset signal)
USB reset flag of USBRST register is sticky so during the reset signal this flag can not be cleared by MCU. After finishing the reset register this flag can be cleared by MCU.

**NOTES:**
1. USB reset signal does not effect the MCU's register or state it only affect to USB part (USB module).
2. The effect of USB reset signal and hardware reset is same except for USBRST register.
3. If you power on the USB controller the USB reset flag is set so you should clear this flag in your initial routine.

## STALL CONDITION

Application: KS86P/C610x4, KS86P/C6504, KS86P/C6408.

Condition:

1. In during the a Data stage a command pipe sent more data or is requested to return more data than was indicated in the Setup stage, it should return STALL.(8.5.2.1)
2. In BULK transfer if the endpoint was halted, STALL is returned to indicate that the host should not retry the transmission because there is an error on the function.(8.5.1)
3. When a request is received by a device that is not defined for the device, is inappropriate for the current setting of the device, or has values that are note compatible with the request, then a Request Error exists. The device deals with the Request Error by returning a STALL PID in response to the next Data stage transaction or in the Status stage of the message.(9.2.7)
4. If an unsupported or invalid request is made to a USB device, the device responds by returning STALL in the Data or Status stage of the request. If the device detects the error in the Setup stage, it is preferred that the device returns STALL at the earlier of the Data of Status stage.(9.4)
5. Control pipes have the unique ability to return a STALL handshake due to function problems in control transfer.

Related flag:

1. EP0CSR.2 (Sent Stall)
   This flag set by hardware when the USB module send stall packet to host.
2. EP0CSR.5 (Send Stall)
   This flag set by MCU if the user want to send stall packet to host. This flag automatically cleared by hardware when send one stall packet to host.
3. EP1CSR.1 (Force Stall)
   This flag set by MCU if the user want to send stall packet to host. If this flag is set by user then USB module continuously send stall packet to host until cleared by MCU.

By setting the flag:

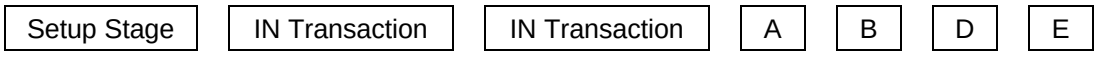
1. Endpoint 0 (Control endpoint)
   If there is problem and MCU want to send stall during control transfer then MCU should set send stall flag for **one time send stall packet**. You must notice after send one stall packet this flag cleared by hardware. (See example below)
2. Endpoint 1 (interrupt endpoint)
   If MCU want to send stall packet during interrupt transfer then MCU should set this flag. If this flag is set the USB controller send stall packet continuously. If MCU clear this flag then USB controller stop send stall packet. **The hardware does not clear this flag automatically so MCU should clear this flag after solve the stall condition.**

By hardware: In control transfer, if there is protocol violation then the hardware send stall packet to host automatically and set sent stall flag for noticing MCU. Basically the blow case, the USB controller send stall packet by hardware.

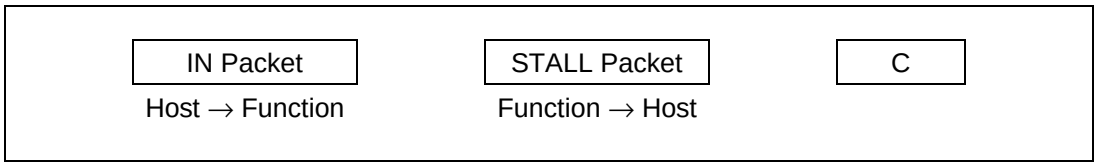
CASE 1: Host send IN packet without Setup stage.
CASE 2: Host send OUT packet without Setup stage.

**% If the USB controller send stall packet to host then the control transfer is finished. If host send in packet then the hardware treat this is protocol violation. (see example)**

Example: Control transfer: The host send IN or OUT packet to function after receiving stall packet.




In A point MCU set send stall flag then the B part is as below;



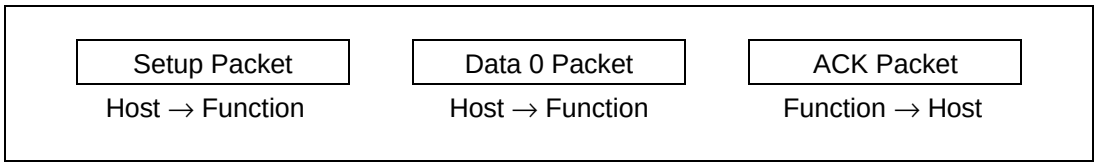

**B Part**

In C point the hardware clear send stall flag and set sent flag. If the function send stall packet to host then the control transfer in finished without status stage so the function expect to receive setup stage.

% Sent flag does not affect to any state of USB controller. The role of this flag is just notice to MCU. Clear this flag or not has no meaning to USB controller.


**D Part**

In D part the USB controller send stall packet because the control transfer is finished so sending IN packet without setup stage is protocol violation. The hardware send stall packet and set sent stall flag continuously until receiving setup packet.




**E Part**

After E part the stall condition is solved.

% If you want to set send stall flag then you should set this flag before clearing OUT_PKT_RDY flag.

**EP1CSR USAGE AND IMPLEMENTATION.**

Application: KS86P/C6104, KS86P/C6504, KS86P/C6408.

Direction: Endpoint 1 only support IN (Function → Host) interrupt transfer.

Contents:

| | |
|---|---|
| Bit 7 | Clear Data Toggle. |
| Bit 6 - Bit 3 | MAX Packet |
| Bit 2 | Flush FIFO |
| Bit 1 | Force Stall |
| Bit 0 | In_Pkt_Rdy |

Usage:

(1) Clear Data Toggle.
Write "1" to this flag : Initialize data toggle sequence. Hardware clear this flag after initializing the toggle sequence.

**NOTE:** This flag was prepared for accident. Usually not use. Until now I use this flag only for test. The data toggle start data0 after reset.

(2) Max Packet.
This field use when the user want to limit the max packet. The reset value "1000" so the maximum packet size is 8 bytes. The MCU can not write more data then this value. This value does not define the sending data size, it only define the maximum size. If the MCU write only 3 bytes then the function send only three bytes. USB module have counter which count the number of written data to FIFO.

example 1)
```
NOP; Assume internal counter is "0".
     ; And MAXP field is "1000".
ld FIFO, data; internal counter become "1".
ld FIFO, data ; internal counter become "2".
ld FIFO, data ; internal counter become "3".
Set in_pkt_rdy ;
; After set in_pkt_rdy the MCU should not
; write data until it is cleared. This scheme can
; Solve the dual access problem.
; Internal counter is 3 so the function send 3 bytes
; to host when the function received IN packet.
```

example 2)
```
NOP; Assume internal counter is "0".
     ; And MAXP field is "0100".
ld FIFO, data; internal counter become "1".
ld FIFO, data; internal counter become "2".
ld FIFO, data; internal counter become "3".
ld FIFO, data; internal counter become "4".
ld FIFO, data; internal counter stay "4".
; This instruction does not work because the MAXP
; field is 4.
Set in_pkt_rdy;
; Internal counter is 4 so the function send 4 bytes
; to host when the function received IN packet.
```

(3) Flush FIFO.
This field use when the user want to clear the contents of FIFO. If the user write "1" to this flag than clear the whole and initialize the internal counter and generate interrupt signal to MCU. See the example.

example)
```
NOP; Assume internal counter is "0".
ld FIFO, data ; internal counter become "1".
ld FIFO, data ; internal counter become "2".
ld FIFO, data ; internal counter become "3".
ld FIFO, data ; internal counter become "4".
Set FLUSH_FIFO ; internal counter become "0".
; Clear in_pkt_rdy if this bit was set.
; Generate interrupt signal to MCU.
; And cleared by hardware.
```

(4) Force Stall.
If this flag is set than the function send STALL packet when receive IN packet. This flag should be clear by MCU. This is not cleared by hardware. See Technical note 1.2.

(5) In_Pkt_Rdy.
After loading data to FIFO than this flag should be set. See below examples.

example)
```
ld FIFO, data; Assume in_pkt_rdy is "0".
• • •
ld FIFO, data; Finish the loading.
; If the function receive IN packet then send NAK
because the value of in_pkt_rdy is "0".
Set in_pkt_rdy;
; If the function receive IN packet then send data
; because the value in_pkt_rdy.
; The value of in_pkt_rdy stay "1".
Receive the ACK from host;
; The value of in_pkt_rdy become to "0". This is
; done by hardware.
; And generate interrupt signal t MCU.
```

**BOUNDARY CONDITION PROBLEM.**

Application: KS86P/C6104, KS86P/C6504, KS86P/C6408.

Direction: Boundary condition arise from IN (DEV → HOST) direction.

Condition : 1) Happen in Control transfer.
2) Host want the data exactly the multiple of MAX Packet number.

ex) Assume the max packet number is 8 and host want 2 bytes.

Problem: SPEC does not define Exact protocol in this case.
case 1, 1) Host send setup transaction.
2) Host send IN packet and Device reply 8 bytes.
3) Host send IN packet and Device reply 8 bytes (16 bytes).
4) Host send IN packet and Device reply 8 bytes (24 bytes).
5) Host send OUT packet for status stage.

case 2, 1) Host send setup transaction.
2) Host send IN packet and Device reply 8 bytes.
3) Host send IN packet and Device reply 8 bytes (16 bytes).
4) Host send IN packet and Device reply 8 bytes (24 bytes).
5) Host send IN packet and Device reply zero-length packet.
6) Host send OUT packet for status stage.

Solution: Samsung USB controller should reply both cases in one firmware.
Samsung USB controller can work with two cases like below.

(1) Usage : in_pkt_rdy, data_end flag.
(2) MCU should set these two flag between 4) and 5) in above cases.

example) 1. Setup transaction. Load data to FIFO and set in_pkt_rdy..
2. IN-Data1-ACK, Generate interrupt. Load second data to FIFO.
And set in_pkt_rdy. (Send 8 bytes, load 8 bytes).
3. IN-Data0-ACK, Generate interrupt. Load third data to FIFO.
And set in_pkt_rdy. (Send 16 bytes, load 8 bytes).
4. IN-Data1-ACK, Generate interrupt. (Send 24 bytes).
MCU should set in_pkt_rdy and data_end flag at the same time.
5. If host send IN packet then device send zero-length packet.
And do the status stage.
If host send OUT packet then device send ACK.

**NOTE:** If MCU does set in_pkt_rdy in 4 of example then device send STALL when received another IN Packet.

## USB RST FLAG USAGE.

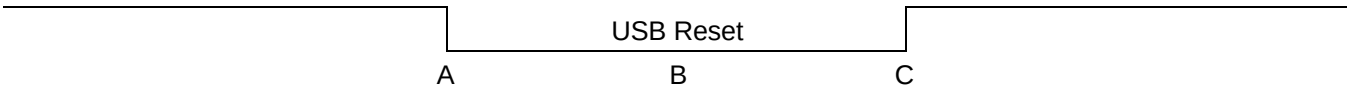
Application: KS86P/C6104, KS86P/C6504, KS86P/C6408.

Purpose: This flag made for just monitoring.

Issue: This flag use USB_RSTN signal (This signal come from internal USB module.) as a synchronous set signal. When power on the internal state going to known value. Since this transient value this flag set at starting. So MCU should clear this flag before starting the program.

**NOTE:** This flag is use just for monitor. Basically the Controller itself does not affect from USB reset signal. But some condition this might be needed for some reason.

Usage:




1) usbrst flag set from the point 2.5us from A point.
2) This flag cannot clear by MCU between A point and C point.

**NOTE:**
```
ld R0, usbrst ; If R0 is 01h then USB module receive reset
              ; signal now or before. Means present state is
              ; in after A point.
ld usbrst, #00h;
ld R0, usbrst ; If R0 is 01h then present state is in B region.
              ; If R0 is 00h then present state is after C point.
```

3) This flag can be cleared by MCU after C point.
4) This flag can be writable before A and after C point.

**NOTE:** Can write "1" or "0" if present state is not reset region.

**SUSPEND / RESUME SUPPORTING.**

Application: KS86P/C6104, KS86P/C6504, KS86P/C6408.

Purpose: Supporting USB suspend SPEC.

Implementation: USB module have a 3 ms counter for this purpose. This counter cleared when USB traffic happen (Including Keep Alive signal) . So this is a idle counter. If there is no USB traffic during 3 ms then generate a suspend interrupt. If MCU receive this interrupt then do a suspend sequence.
Samsung USB controller have three mode, these are normal, idle, stop. In suspend mode Samsung MCU usually using stop mode, In stop mode the oscillator is down, so there is no clock. Samsung MCU use D+, D- signal as a clock source when suspend mode. So Samsung can detect resume signal without clock and generate a resume interrupt. When MCU receive interrupt signal in stop mode, then the oscillator wake up and go interrupt service routine.

Usage1: When receive resume signal from upstream port.

```
                initialization routine;
                ld usbint, #0ffh; Enable interrupt.
                • • •
        int     nop; Here is interrupt service routine label.
                check interrupt vector; Assume this interrupt is suspend.
                jp suspend;

        suspend nop;
                do suspend routine;
                stop; This means stop oscillator for saving power
                    ; The whole chip action stop.
```

This stop can be recovered from external interrupt (external resume, keystroke, mouse moving, etc.) source. That means if MCU receive interrupt then the oscillator automatically activating and jump to interrupt service routine. In resume case, D- line connected to the clock port of resume detect F/F, so any change from idle mode (idle to reset, idle to K-state) can be detected by this circuit. If this F/F detected action then generate a interrupt to MCU part and going to interrupt routine.

```
        int     nop; If the MCU receive resume or external interrupt then the PC
                   ; get this address.
                check interrupt vector; Assume this interrupt is resume.
                jp resume ; Jump to resume interrupt service routine.

        resume  nop;
                do wakeup process;
                jp main; Jump to main loop;
```

```
Usage2:         When USB controller drive resume signal to upstream port.
                        Assume suspend mode;
                        Receive mouse moving or keystroke;
                int     nop; interrupt service routine.
                        check interrupt source; Assume the interrupt source is mouse
                                                ; moving or keystroke
                        jp send_resume; Jump to resume service routine.

send_resume nop;
                ld PWRMGR, #02h ; Start to send resume.
                ld R1, #some_value; Spec says the length of resume signal should
                                ; between 1 ms to 15 ms, so the programmer
                                ; should select adequate value.
loop3           nop;
                dec R1;
                cp R1, #00h;
                jp nz, loop3;
                ld PWRMGR, #00h; End of resume.
                jp main;
```

**RESPONDING BEFORE RESET SIGNAL.**

Application: KS86P/C6504, KS86P/C6408, KS86P/C6308

Problem: Samsung USB controller respond to host before reset signal.

Solution: Basically Samsung USB controller respond to host before reset signal. This violation can be solved by heuristic method which considering transceiver characteristic. Above Samsung USB controller can support PS2/USB application. Samsung USB controller share D+/PS_CLK, D-/PS_DATA pins and can change the port mode. So, before USB reset signal we can use these pins as PS2 input (with pullup in D- pin) mode. And after receive the USB reset signal MCU can change these pins mode to normal USB D+, D-.
MCU can read these pins using PS2 input mode. And USB reset signal length is at least 10 ms. MCU can figure it out whether reset or not.

Usage:
```
initialization code;
ld usbsel, #00h; Set port mode as a PS2 mode.
ld P2CONINT, #10h; <= KS86P/C6504 case
                   ; Set D+ as schmitt trigger input.
                   ; Set D- as schmitt trigger input with pull-up.
                   ; This means USB connection established.
loop1   ld R0, P2; Read port data.
        cp R0, #00h; Check port data.
        jp nz, loop1 ; If the port data is not zero then go to loop1.
                     ; This loop wait until receive the reset signal.
        ld R1, #03h ; Check three iteration for robustness.
                    ; The length of USB signal is from 2.5 us to 10 ms,
                    ; so 2,3,4 or 5 times checking is enough.
loop2   ld R0, P2;
        cp R0, #00h;
        jp nz, loop1; If the SE0 is too short then this signal is not USB reset
        signal, so return to loop1.
        dec R1;
        cp R1, #00h;
        jp nz, loop2;
        ld usbsel, #01h; Change from PS2 mode to USB mode.
        do normal operation ;
```

**NOTE:** If MCU set transceiver as PS2 mode then the input signal does not go through to USB module, so USB module does not respond to host.

## INTERNAL HARDWARE STRUCTURE OF SAMSUNG USB CONTROLLER.

Application: KS86P/C6504, KS86P/C6408. KS86P/C6104

1. When the device receive setup transaction with DATA1 packet.
   Answer) When the device receive setup transaction with data1 packet then the device send ACK packet and does not generate interrupt signal. That means the device ignore this transaction since the data pid is not proper.
2. When the device receive setup transaction with data0 packet which have non-8 byte data length.
   Answer) The device respond with ACK and generate interrupt to MCU. If the device read EP0CNT register then the device can get the data number of this packet.
3. When the device receive setup transaction which have more then 8 bytes data packet.
   Answer) The device keep first 8bytes data and response with ACK packet. And the EP0CNT have the value 8. That means if the device receive more than 8 bytes then there is no way to distinguish this one from 8 bytes setup transaction. I upload this issue to USB-IF web board and I received positive message.
4. Device Address scheme.
   Answer) Device address is generated by selecting two source. The below is the fragmentation of verilog file. Verilog is a hardware description language so the hardware function is same with this statement. I synthesized the hardware using this program.
   When I designed this one I only think about first set address command but now we must consider the second set address command. Fortunately we can solve this issue using firmware.
   Anyway you can figure it out what's going on internal hardware.

   ```
   assign fun_addr = (device_configured) ? uc_fun_addr_reg: 7'h00;
   /* This means if the device_configured is 1 then the value of fun_addr is same to
   uc_fun_addr_reg. And if the device_configured is 0 then the value of fun_addr is zero. */

   /* Device_configured signal used for detecting the status stage of set address command.
   Before status stage this value is 0 and after status stage this value going to 1. */
   ```

```
always @(posedge clk or negedge reset) begin
        if (reset) begin
           device_configured <= 1'b0;
        end
        else begin
           /* The below is the setting condition of the device_configured signal. After setting this
           value this signal remain 1 until receive reset signal. */
           device_configured <= ( | (uc_fun_addr_reg) & endpt0_data_end
                  & endpt0_clr_data_end & ~endpt0_setup_end
                  & ~endpt0_set_setup_end) | device_configured;
           /* Set condition of this signal.
             1. uc_fun_addr_reg have non-zero value.
             2. endpt0_data_end should be 1. (this flag is CSR0[3]).
             3. endpt0_clr_data_end should be 1. (This means end of status stage.)
             4. endpt0_setup_end should be 0. (This means not terminated by another setup
                transaction.)
             5. endpt0_set_setup_end should be 1. (This signal is a setting signal of
                endpt0_setup_end.)
           */
        end
end


always @ (posedge clk or negedge reset) begin
        if (~reset) begin
             uc_fun_addr_reg <= 7'b00;

        end
        else begin
             /* The below statement means the firmware can change the
             uc_fun_addr_reg anytime. */
             if (uc_wrt & uc_dec_fun_addr)
                uc_fun_addr_reg <= uc_data[6:0] ;
        end
end
```

## SET ADDRESS COMMAND SUPPORTING

Application: KS86P/C6104, KS86P/C6504, KS86P/C6408.

Purpose: Supporting multiple set address command and robustness.

Implementation: Basically above USB controller does not support multiple set address command. But above USB controller can support multiple set address and robustness(missing status stage) using some heuristic method. The sequence of firmware is as below and the flowchart of this scheme is in next page. The below scheme perfectly supports missing status stage situation and multiple set address command using one method.

Scheme.

1. Prepare variable PrevAddr(8 bits), DevAddr(8 bits) and initialize as below.
   PrevAddr = 8'hFF; DevAddr = 8'h00; go to 2;
2. Wait until receive interrupt.
   if (received interrupt == Endpoint 0 interrupt) then go to 3
   else go to 2.
3. If (received command == set address) then go to 4
   else execute this command and go to 2.
4. Write PrevAddr to device address register;
   Write 8'h48 to CSR0;
   DevAddr = received address; go to 5;
5. Wait until interrupt.
   if (received interrupt == status stage) then
   Write DevAddr to device address register;
   PrevAddr = DevAddr; go to 2;
   else go to 6 ;
6. This state exists for missing status stage situation.
   if (PrevAddr == 8'hFF) then
   Write 8'h00 to device address register.
   revAddr = 8'h00;
   go to 2.
   else go to 2.

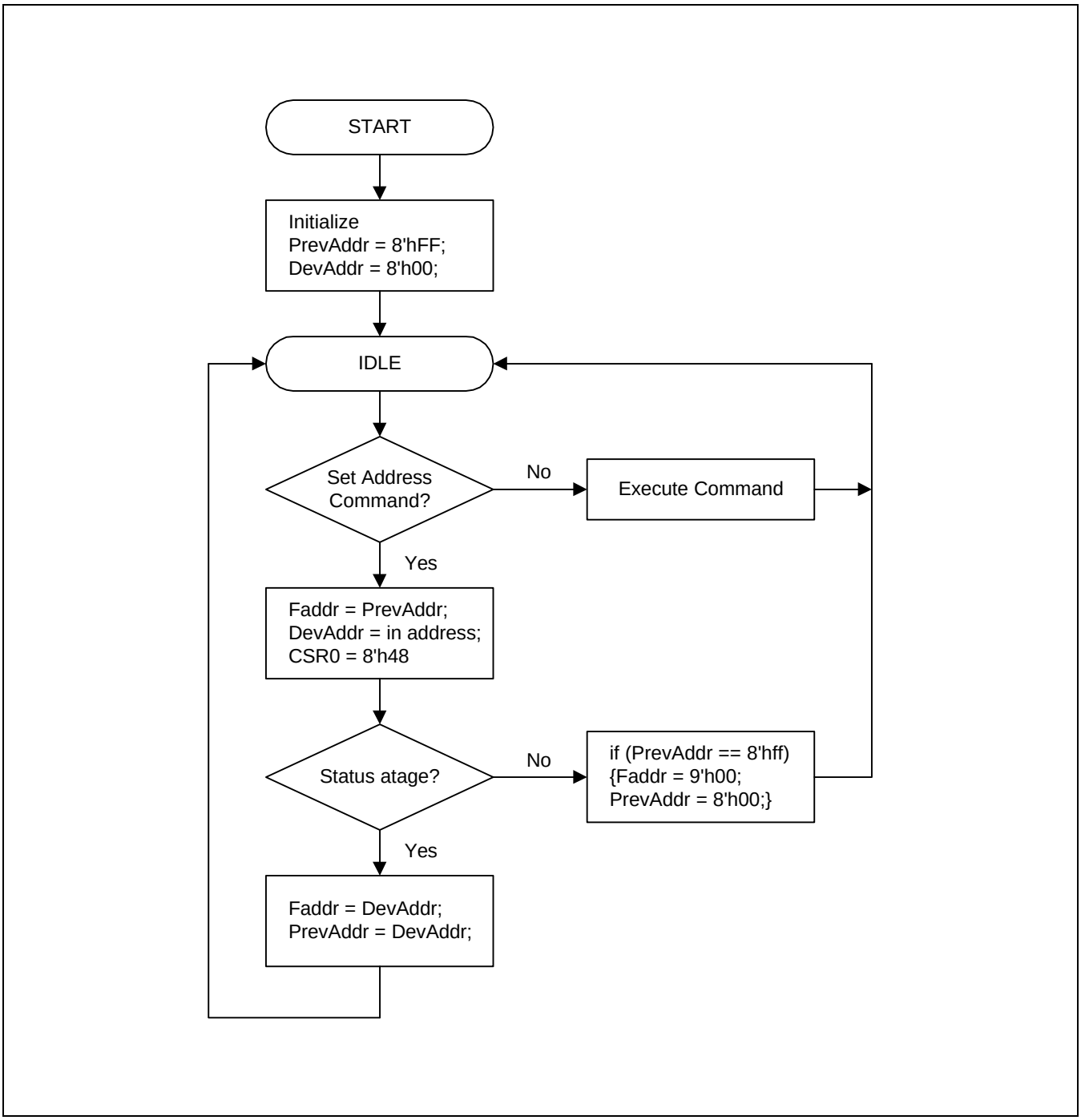



**Figure 19-1. Set Address Command Supporting**

## DECODING TIP OF EP0 CONTROL REGISTER VALUE

Application: KS86P/C6104, KS86P/C6504, KS86P/C6408.

Purpose: Robust decoding scheme of endpoint control status register value.

Implementation: This part will cover robust way of implementing endpoint 0 CSR value. The first one is a description which describe the flow of the implementation. The next one is a flowchart of description.
The below description will help you to implement authentic method of endpoint 0 CSR as a interrupt service routine. The below scheme assume you use event-driven scheme not polling scheme. If you use polling scheme then you may get another value of ep0csr. I will explain these issues briefly in last part.

### Scheme example

1. IDLE state (Busy waiting & if interrupt occurred then go to step 2).
2. Read usbpnd register for checking interrupt source.
   If the interrupt source is not endpoint 0 then
   execute appropriate action and go to step 1.
   else go to 3.
3. Check state variable which initialized to IDLE.
   If state is set_addr then begin
   if (CSR0 == 00h) then begin
   LD Faddr, DevAddr.
   LD PrevAddr, DevAddr.
   go to step 1.
   end
   else begin
   if (PrevAddr == FFh) then begin
   LD Faddr, 00h.
   LD PrevAddr, 00h.
   go to step 4.
   end
   else go to step 4.
   end
   end /* This is for supporting multiple set address command. */ See Ref, 1.9
   else go to 4.
4. Read ep0csr and store to CSR0 variable.
   if CSR0[4](Setup_End flag) is set then
   assign state = IDLE.
   initialize some variable for preparing next control transfer.
   assign CSR0[4] = 0.
   go to step 5.
   else go to step 5.

```
5.  if CSR0[2](Sent_Stall) is set then
        execute appropriate action.
        assign CSR0[2] = 0.
        go to step 6.
    else go to step 6
6.  If (CSR0 == 8'h01h) then begin
        /*This means this interrupt is come from Setup transaction or Out transaction.*/
        If (state == IDLE) then begin
                Unload EP0FIFO value and decode this command.
                go to step 7.
        end
        else begin
                If (state == OUT) then begin
                        unload EP0FIFO data and check the number of data byte.
                        if the number of data byte is MAX_VALUE then assign ep0csr = 8'h40.
                        else ep0csr = 8'h48.
                        go to step 1.
                else ERROR!!!
        end
    end
    else go to 11.


7.  If this is NO DATA STAGE command  then go to step 8.
    else go to step 9.
8.  If this is set address command then begin
        LD Faddr, PrevAddr.
        LD DevAddr, in_address.
        LD ep0csr, 48h.
        LD state, set_addr.
        go to step 1.
    end
    else begin
        Execute this command.
        go to step 1.
    end
9.  If this command is IN Control transfer then begin
        LD ep0csr, 40h.
        LD state, IN.
        prepare variable for this transfer.
        go to 13(EP0FIFO loading part).
    end
    else go to step 10.
```

10. If this command is OUT control transfer then begin
        if (Transfer is finished) LD ep0csr, 48h.
        else LD ep0csr, 40h.
        go to step 1.
    end
    else begin
        If this command is not supported by MCU then
        LD ep0csr, 60h,
        go to step 1.
    end

11. If (CSR0 == 8'h08) then
        Do nothing and go to step 1.
        /*You can get this value after set 0Ah to ep0csr and receive last in packet
        from host then the MCU send zero-length data packet and generate interrupt.*/
    else go to step 12.

12. If (CSR0 == 8'h00) then
        /*This means the interrupt occurred from status transaction.*/
        if (state == IN) then
                go to 13(EP0FIFO loading part).
        else if (state == IDLE) then
                go to step 1.
        else ERROR.
    else ERROR. (other value is impossible)

13. /* Variable Definition. */
    /* TotDB : Actual length of data. */
    /* ReqDB : Requested data length. */
    /* PreDB : Present data count. */
    /* NowCnt : Present load count.*/
    LD NowCnt, #00h.
    go to 14.

14. If ((PreDB == ReqDB) || (PreDB == TotDB)) then begin
        LD state, IDLE.
        LD ep0csr, 0Ah.
        go to step 1.
    end
    else go to 15.

15. LD EP0FIFO, data.
    INC NowCnt.
    INC PreDB.
    If (NowCnt == MAX_Packet) then
        LD ep0csr, #02h.
        go to step1.
    else go to step 14.

## Flowchart Diagram

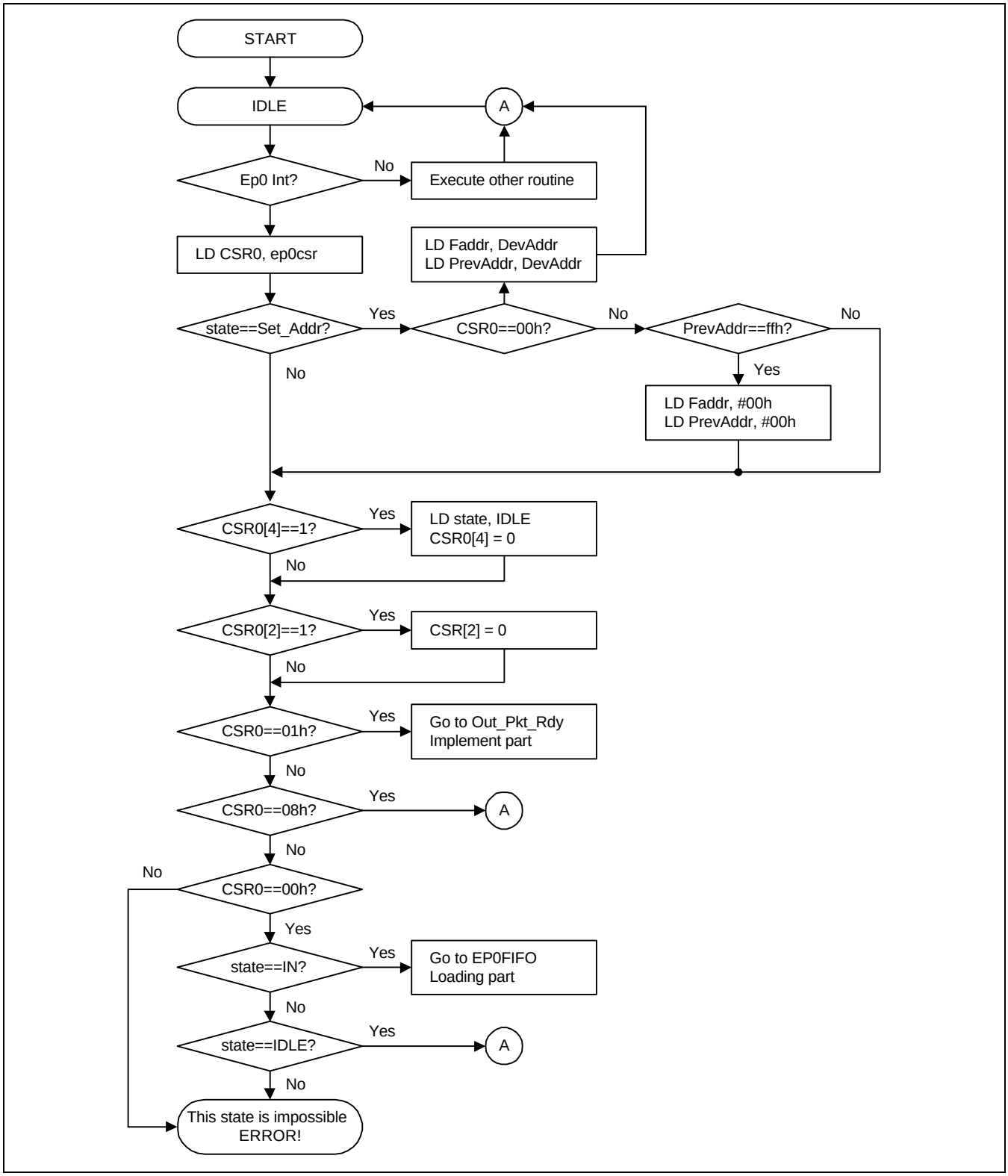



**Figure 19-2. Overall Part**

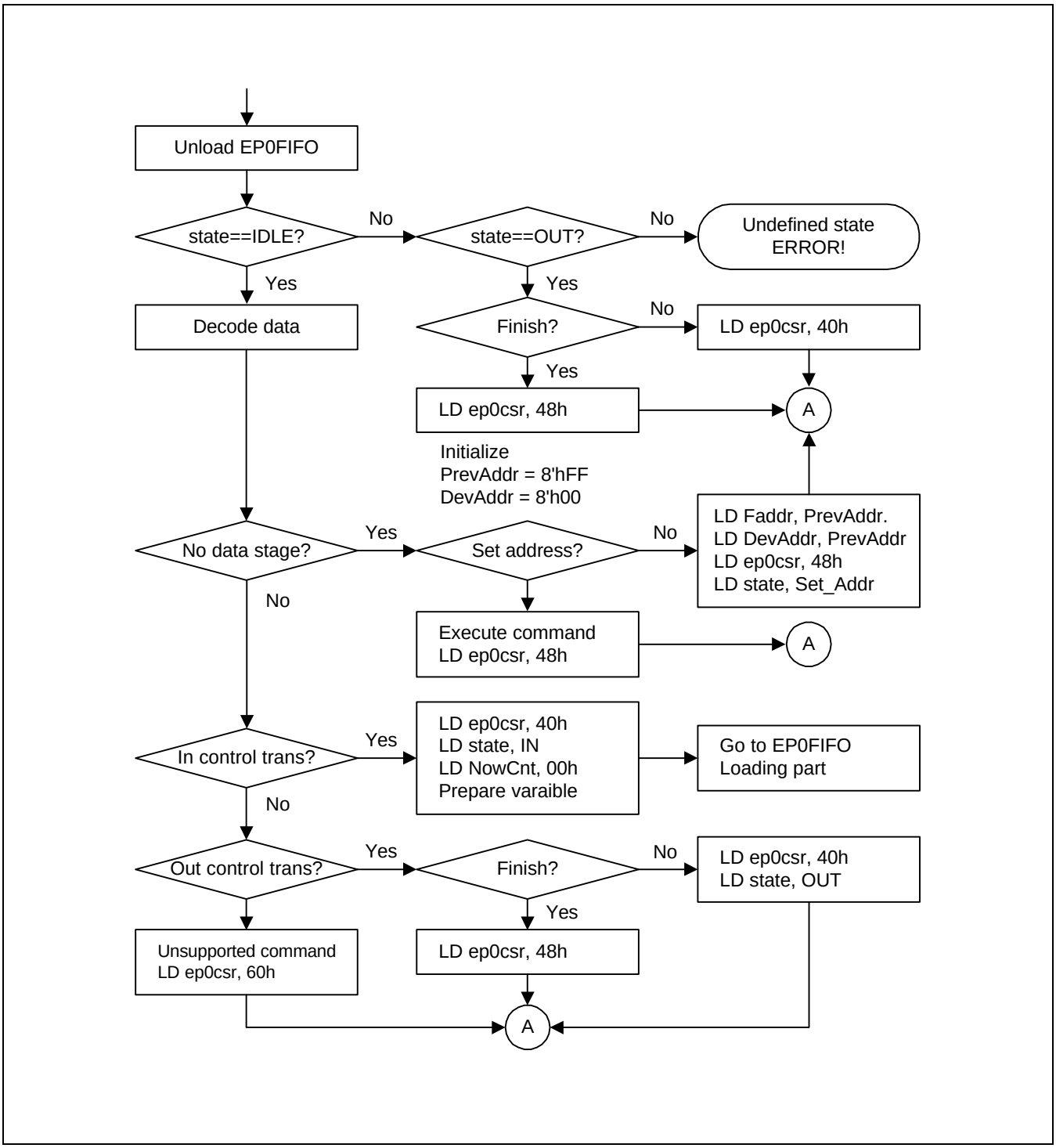
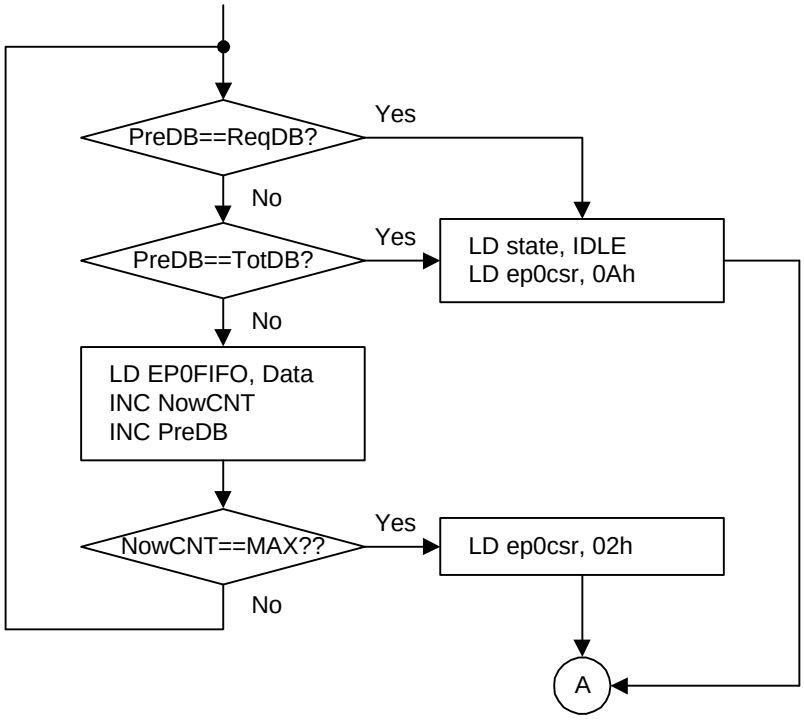



**Figure 19-3. Out_Pkt_Rdy Implement Part**




**NOTE:** TotDB: Actual length of data
ReqDB: Requested length of data
PreDB: Present value of data count
NowCNT: Present FIFO load data count

**Figure 19-4. EP0FIFO Loading Part**

**EP0CSR Flag Definition**

1. Clr_Setup_End.
   1.1. Write only flag. If you read this value then you will get always "0".
   1.2. The MCU use this flag for clearing Setup_End flag. The hardware will automatically clear this flag after clear Setup_End flag.
   1.3. This flag can not be a interrupt source.
2. Clr_Out_Pkt_Rdy.
   2.1. Write only flag. If you read this value then you will get always "0".
   2.2. The MCU use this flag for clearing Out_Pkt_Rdy flag. The hardware will automatically clear this flag after clearing Out_Pkt_Rdy.
   2.3. This flag can not be a interrupt source.
3. Send_Stall.
   3.1. Write only flag. The hardware will automatically clear after setting up send stall condition.
   3.2. If you want to send stall to upstream then you should set this flag. After setting the FSM (For send stall in next transaction) this flag cleared by hardware.
   3.3. Actually this flag can not be a direct source of interrupt but can be a second source of interrupt. After sent stall packet the SIE will set Sent_Stall flag and generate interrupt to MCU.
4. Setup_End.
   4.1. Read only flag. This flag indicate the termination of control transfer. If the MCU set Data_End flag and do a successful status stage then the control transfer is terminated by normally. In this case this flag will not be set. That means this flag will be set when the control transfer is terminated without Data_End setting or terminated by new setup transaction or protocol violation(In this case, the SIE will send stall and set Sent_Stall flag). You will get "1" value in above situation when interrupt occurred.
   4.2. If you get "1" value when you get interrupt then you should clear this flag using Clr_Setup_End flag and should initialize state variable for next new control transfer. According to protocol, the next transaction should be a setup transaction.
   4.3. This flag is source of interrupt. When this flag do a transaction from "0" to "1", the SIE will generate a interrupt to MCU.
5. Data_End.
   5.1. Read/Write flag. But automatically cleared by hardware after sending status packet in status stage.
   5.2. Setting this flag means that MCU will only accepts status stage. So if the controller receive another transaction then the MCU will send stall packet (except for setup transaction). If the MCU receive new setup transaction then the hardware will clear this flag and set Setup_End flag.
   5.3. This can be a source of interrupt. When this flag do a transition from "1" to "0", the SIE will generate a interrupt to MCU.

6. Sent_Stall.
   6.1. Read/Clear flag. If MCU receive interrupt and get this value set then MCU should clear this flag through writing "0" to this flag.
   6.2. If the SIE sent stall to upstream then this flag will be set. There are two source which send stall to upstream, The one is set Send_Stall flag by MCU and the other case is by SIE automatically when the host do a protocol violation. The below is the example of protocol violation which the SIE send stall automatically.

   example)
   A. If the host send IN or OUT token without SETUP transaction.
   B. If the host send IN status transaction on IN control transfer.
   C. If the host send OUT status transaction on OUT control transfer.
   D. If the host send OUT status transaction on non-data transfer.

   6.3. This can be a source of interrupt. When this flag do a transition from "0" to "1", the SIE will generate a interrupt to MCU.

7. In_Pkt_Rdy.
   7.1. Read/Write flag. This value can be set by MCU but cleared by hardware after successful IN transaction.
   7.2. After loading the data to endpoint 0 FIFO the MCU should set this flag. The MCU can not write data to endpoint 0 FIFO during this flag is set. Since set this flag means that data loading is finished. And if the controller receive IN token then send these data to upstream. And if this value is "0" when received IN token then the SIE will send NAK packet to upstream. If MCU set this flag without data loading then the SIE will send zero-length data packet to upstream. If the MCU did not receive ACK packet after sending loaded data then this value stay "1" and will retry in next IN transaction.
   7.3. This can be a source of interrupt. When this flag do a transition from "1" to "0", the SIE will generate a interrupt to MCU.

8. Out_Pkt_Rdy.
   8.1. Read only flag. This value can be set by SIE but cleared by MCU through setting Clr_Out_Pkt_Rdy flag.
   8.2. If the controller receive SETUP transaction in idle state and receive OUT transaction during data stage of OUT control transfer then the SIE set this flag and generate interrupt to MCU. If the MCU get this value set then should download the data to local buffer. The MCU can not write data to endpoint 0 FIFO if this value is set. So if MCU want to write data to endpoint 0 FIFO then should clear this flag before loading. If the controller receive IN or OUT transaction during this value set then the SIE send NAK to upstream. But if the MCU receive SETUP transaction then the SIE set Setup_End flag and set this flag and generate interrupt again.
   8.3. This can be a source of interrupt. When this value do a transition from "0" to "1" the SIE generate a interrupt to MCU.

9. Interrupt source.
   9.1. Data_End flag : "1" → "0".
   9.2. Sent_Stall : "0" → "1".
   9.3. In_Pkt_Rdy : "1" → "0".
   9.4 Out_PKt_Rdy : "0" → "1".

10. Possible value of ep0csr when you get interrupt and appropriate action.
   ¡Ø The first two flag (Clr_Setup_End, Clr_Out_Pkt_Rdy) are always "0" value when you read so I will explain just low 6 bits.
   ¡Ø If the MCU did not clear Sent_Stall flag then this value remain "1" until cleared. So Sent_Stall can be set to any combination with below.
   10.1. "000001" : Set Out_Pkt_Rdy.
   If you get this value then you should check your state variable. If the state is in OUT control transfer then this interrupt notice the receiving of OUT data transfer. So you should download the FIFO data and clear this flag and do a appropriate action. If the state is in IDLE then this interrupt means that the controller receive new setup transaction so download the FIFO data and should decode and do a appropriate action.
   10.2. "000100" : Set Sent_Stall.
   If the controller sent stall to upstream then this value be set by SIE. This flag is for just monitoring. So just clearing this flag is enough action.
   10.3. "001000" : Set Data_End flag.
   Actually this flag does not a generate interrupt condition. When you set In_Pkt_Rdy and Data_End flag at the same time and the controller receive IN transaction and send data and receive ACK packet then only In_Pkt_Rdy flag is cleared and generate interrupt. In this case you can get this value set but there is no meaning so just leave these alone is enough action.
   10.4. "010000" : Set Setup_End.
   If the former control transfer is terminated by abnormally then this flag is set by hardware. So if MCU got this value then should clear this flag and initialize some variable and should prepare future control transfer.
   10.5. "010001" : Set Setup_End and Out_Pkt_Rdy.
   If the former transfer is terminated by new SETUP transaction then MCU can get this value. If the MCU get this value then should initialize variable and reset the status variable according to new command.
   10.6. "000000" : Nothing is set.
   This value can get in two case. If the former control transfer is terminated with Data_End setting. The other case is happen in data stage of IN control transfer. In IN control transfer if you load data to endpoint 0 FIFO and set In_Pkt_Rdy and the controller receive IN token and send data and receive ACK packet then the SIE clear In_Pkt_Rdy and generate interrupt.

# NOTES