LS2051 (also refer to LS4051) is high performance eight bit microcomputer designed and manufactured by our company and is compatible with MCS-51 instruction set, and its internal function, pin function, pin configuration, and electrical characteristics of pins are compatible with AT89C2051. LS2051 supports simultaneous operation of two independent or associated programs. The performance when only one program is executed is 1.27 times of that of AT89C2051, and the processing capacity is 2.55 times of AT89C2051 when two programs are executed simultaneously.

Features

- Compatible with MCS-51 instruction set，pins are compatible with AT89C2051.
- 2/4KB internal flash program memory, Endurance: 2000 times erase/write cycles.
- 3.0V to 5.5V operating range.
- 0Hz to 24 MHz frequencies.
- Two-level of Encryption of Program Memory.
- 128×8B internal SRAM.
- 15 programmable I/O ports，20mA sink current，drives LED directly.
- Six interrupt sources.
- Two 16-bit timer and counter.
- One programmable UART.
- SPI programming interface.
- One high-resolution on-chip Analog Comparator.
- Low power idle and power-down modes.

Description

LS2051 contains 2K bytes of program memory (LS4051 contains 4K bytes), 128 byte of data memory (SRAM), two 16-bit timer/counter, one five-vector secondary interrupt architecture , a single-kernel for execute two programs, fifteen IO ports, a full duplex serial port, a high-accuracy on-chip analogy comparator, oscillator and clock circuitry.

The operating range of LS2051 is 0Hz to 24MHz, and supports the idle power saving mode and the power-down power saving mode which are software selectable. The idle mode stops the CPU, while allowing the SRAM, timer/counter, serial port and interrupt machine and the power-down system to continue functioning. The power-down mode saves the SRAM contents but freezes oscillator and turns all the internal clocks off.

LS2051 achieves the function of double kernels, that is, it can execute one programming singly, and it can also execute associated or non-associated two programs concurrently. The performance of executing one program is 1.27 times of compatible chips, and that of

processing two programs can reach 2.55 times of compatible chips at most.

Pin Description

Pin configuration and its multiplexing function of LS2051 are as Figure 1.

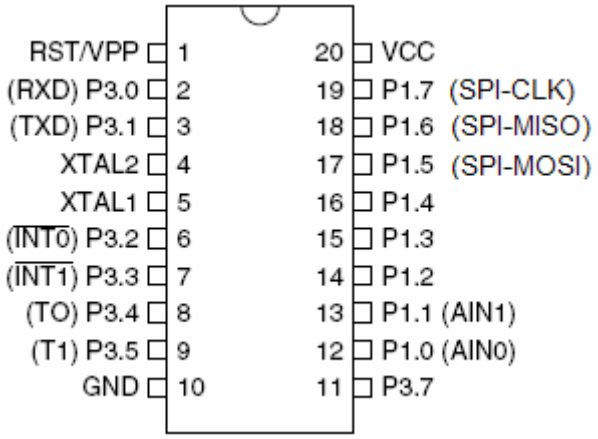


Figure 1: LS2051 Pin Configuration

- VCC：Supply voltage
- GND：Ground
- P1 port: an eight-bit programmable bi-directional I/O port. Pin P1.2 to P1.7 provide internal pull-ups. P1.0 and P1.1 serve as the positive input (AIN0) and the negative input (AIN1) of the on-chip comparator and with no internal pull-up, one can add pull-ups for the pins if needed. P1 port output buffer can absorb 20mA current and can drive LED directly. A pin of P1 can be used as input only after “1” is written into the pin; A pin can output “1” or “0” when “1” or “0” is written into the pin. P1 port is used as inputting port when chip is reset . When pins P1.2 to P1.7 are taken as inputting and are pulled down, they will produce original current because of their inside pull-up resistance (I_{IL}). When programming, P1.5, P1.6 and P1.7 should correspond to programming interface signal, SPI-MOSI, SPI-MISO and SPI-CLK. Note that 4.7K ohm pull-up resistance should be added when the speed of I/O port is over 40 KHz.
- P3 port: P3.0 to P3.5, P3.7 are seven bi-directional I/O pins with internal pull-ups, P3.6 is the output of the on-chip comparator and is not connected outside the LS2051. Port 3’s output buffer can sink 20mA and can drive LED directly. A pin of P3 can be used as input only after “1” is written into the pin; A pin can output “1” or “0” when “1” or “0” is written into the pin. P3 port is used as inputting port when chip is reset. When used as inputs, Port 3 pins are externally being pulled down and will produce current (I_{IL}).

Multiplexing function of P3 port is shown in Table 1. Note that 4.7K ohm pull-up resistance should be added when the speed of I/O port is over 40 KHz.

Port pin	Alternate Functions
P3.0	RXD（serial input port）
P3.1	TXD（serial output port）
P3.2	$\overline{\text{INT0}}$（external interrupt 0）
P3.3	$\overline{\text{INT1}}$（external interrupt 1）
P3.4	T0 （timer 0 input）
P3.5	T1 （timer 1 input）

Table 1: Special Functions of Port P3

- RST: Reset input. High is active, after power-on or power-down mode of chips, chips can execute smoothly only when the reset signal of 1s must be kept over 100us. Under other circumstance, holding the RST pin high for two machines cycle can reset the chip. Each machine cycle takes 12 oscillator or clock cycles.
- XTAL1: Input to the inverting amplifier of the internal oscillator and input to the internal clock generating circuit.
- XTAL2: Output from the inverting amplifier of the internal oscillator.

Clock Characteristics

Either quartz crystal or ceramic resonator may be used, as figure 2. XTAL1 is the input of external clock. When external clock is used, XTAL2 is left unconnected, as Figure 3. There is a divider inside LS2051 to perform the function of dividing-by-two and forming for output clock of internal oscillator or for external input clock from pins. Therefore there is no special requirement for external input clock, but it is must be in accordance with current and alternating standards of clock signal.

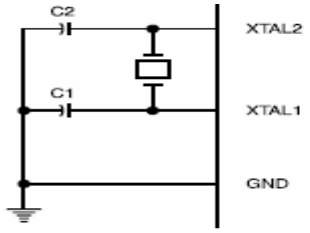


Fig 2: Clock Circuitry Using Internal Oscillator

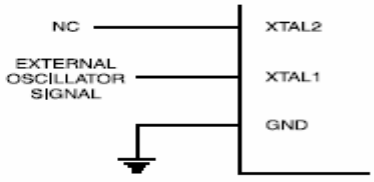


Fig 3: Connection of External Clock

When quartz crystal is adopted： C1,C2=30pF±10pF.

When ceramic resonator is adopted ： C1,C2=40pF±10pF.

Special Function Registers

The memory map for special function register (SFR) is shown in Table 2. The blank bytes are not defined in this version. They will return indeterminate values when being read.

Addr	B0	B1	B2	B3	B4	B5	B6	B7	Addr
0F8H									0FFH
0F0H	B 00000000								0F7H
0E8H									0EFH
0E0H	ACC 00000000								0E7H
0D8H									0DFH
0D0H	PSW 00000000								0D7H
0C8H									0CFH
0C0H									0C7H
0B8H	IP XXX00000								0BFH
0B0H	P3 11111111								0B7H
0A8H	IE 0XX00000								0AFH
0A0H									0A7H
98H	SCON 00000000	SBUF XXXXXXXX							9FH

90H	P1 11111111								97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000			8FH
80H		SP 00000111	DPL 00000000	DPH 00000000				PCON 0XXX0000	87H

Table 2: SFR Map and Reset Values

Note: IE 's d7 bit of Special Register is the interrupt enable bit for the main or first program, and d6 bit is the interrupt enable bit for the second program.

Restriction on Some Instructions

The instruction system of LS2051 is compatible with MCS-51, but one should pay attention to the following two restrictions when using LS2051 to develop application programs.

1、MOVX Related Instructions, Data Memory

The internal data memory of LS2051 is 128 bytes, the depth of stack is 128bytes, accessing to the external data memory is not supported in LS2051, nor is the external program memory. Therefore, no MOVX related instructions should be included in application programs.

A typical 51 assembler will still assemble such instructions even if they read and/or write external memories. Users should know the physical features and limitations of LS2051 and adjust instructions correspondingly.

2、Branching Instructions

The unconditional branching instructions , such as LCALL, LJMP, ACALL, AJMP, SJMP and JMP @A+DPTR, are executed correctly only when the destination branching addresses fall within 000H to 7FF, otherwise programs might go wrong.

The restrictions of conditional branching instructions, such as CJNE, DJNZ, JB, JNB, JC, JNC, JBC, JZ and JNZ, are the same as that of unconditional branching instructions.

Other Restrictions

- The reset time should be over 100us when chips are power-on and/or waked from power-down mode.
- It won't response to an interrupt when the interrupt service for the interrupt is suspended.
- If idle mode is entered during an interrupt is being served, the idle mode can not be waked up by the interrupt.

- Both programs have their own ACC、B、DPTR、PSW and SP, but the main program (the first program) has 32 general registers while the second has 8. It should be pay more attention to the stack operations in the development of the second program.
- The design of the main program is completely the same as that of AT89C2051. If the second program is not used, an LS2051 is totally compatible with AT89C2051.
- The interrupt processing of LS2051 is more active and effective than that of AT89C2051 because of the peculiar second program processing engine of LS2051. If the interrupt for the main program is enabled while the interrupt for the second program is not enabled, the main program will deal with all the interrupts. If the interrupts for both programs are enabled and the second program has not been started to execute or not been designed, both programs may deal with interrupts, this is automatically arranged by the chip itself.
- Once the second program is started to execute, it won't be interrupted.
- When an interrupt comes, LS2051 jumps into the interrupt service routine before the current instruction is done, while 89C2051 is after the current instruction is done.
- Pay more attention to the possible confliction for public resources when two programs are employed.
- The baud rate of LS2051's Serial way 0 is Fosc/24, and the baud rate of serial way 2 is Fosc/64 and Fosc/128, while that in 89C2051 are Fosc/12, Fosc/32 and Fosc/64.
- If P1 port and P3 port are used to input/output in a rate over 40 KHz, the ports should be pulled up with external resistances of 4.7K.

Program Memory Lock Bit

On the chip are two lock bits which can be left un-programmed (U) or can be programmed (P) to obtain program lock, as shown in Figure 3, lock bit can be turned to invalidation only by erasing .

LB1	LB2	Function
U	U	No program lock
P	U	Further the programming of Flash is disabled
P	P	Further the programming of Flash is disabled, also verify is disabled.

Figure 3: Program Lock of LS2051

Work mode

1、Idle mode

Idle state can be entered through program instructions, and can be terminated by any enable interrupt or a hardware reset. While in idle state, CPU stops to work, but allowing SRAM, timer/counter, serial ports and interrupt system to continue to function. P1.0 and P1.1 should be set to “0” if no external pull-ups are used, or set to “1” if external pull-ups are used to decrease further power consumption.

It should be noted that when idle state is terminated by a hardware reset, the chip normally resumes to work after two machine cycles of the reset. During this time, internal SRAM is inhibited to access, but the port pins are not inhibited. To eliminate the possibility of an unexpected write to a port pin when Idle is terminated by reset, the instruction following the one that invokes to enter an idle state should not be to write to a port pin.

2. Power-down mode

A power-down state can be entered through program instructions, and can be terminated by hardware reset. In the power down state, the oscillator is stopped, and the instruction that invokes power down is the last instruction executed. The on-chip SRAM and Special Function Registers retain their values during the power down state. Reset redefines the SFRs but does not change the on-chip SRAM. The reset signal is invalid until VCC is restored to its normal operating level, and the reset signal must be held active long enough to allow the oscillator to restart and stabilize. P1.0 and P1.1 should be set to “0” if no external pull-ups are used, or set to “1” if external pull-ups are used, this can help to decrease power consumption.

Programming the Flash

LS2051 uses SPI protocol to programming the flash. And special programming driver and cables are also provided.

LS2051’s SPI interfaces will not provide read operation to the internal flash in order to strengthen the security of user’s application software.

SPI Programming Agreement

SPI interface of LS2051 includes RST, SCK(P1.7), MOSI(P1.5) and MISO(P1.6) signals , as Figure 5.

Chips are in programmable state when reset input RST is “1”. At this time it can send programming instructions and codes through SCK and MOSI, MOSI is programming fault indication. Chips are in normal operating state when the reset signal is “0”.

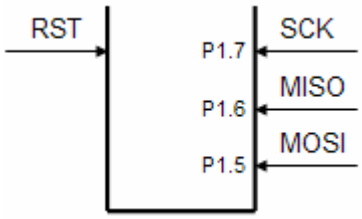


Figure 5: Flash Programming Interface Signals

SPI adopts synchronous serial way to sending byte data, its timing sequence is as Figure 6.

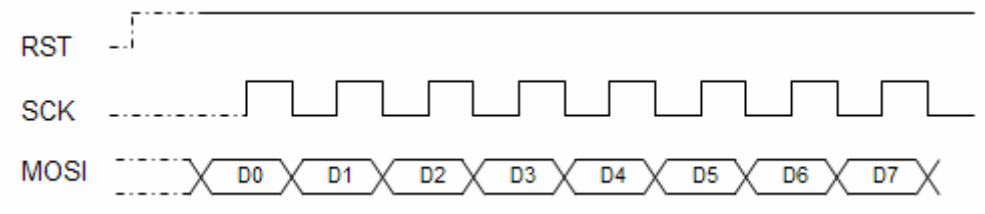


Figure 6: SPI Interface Sequence

When a LS2051 is in programming state, serial data carried by MOSI are sampled at the up edge of SCK, the successive eight bits are set to be one byte, and the seventh bit of a byte is transmitted first.

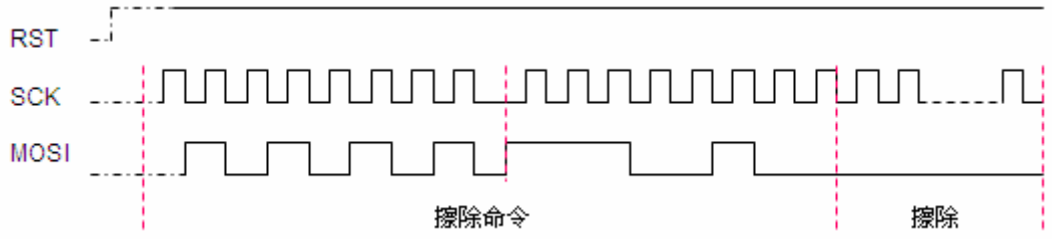
MISO is the verifying output. LS2051 will verify each byte of data. When a byte is failed to pass the verification, MISO will output “1” until next correct verification, and MISO will output “0”.

Programming order

- Write Flash: the sequence of order code is AA-50-AX-AY-data. When SPI interface receives these five bytes successively, chips enter Flash writing operation. The first byte AA and the second byte 50 are order codes, the third byte is high 6-bit address of Flash, the fourth byte is low 6-bit address of Flash, and the fifth byte is data to be input.
- Erase Flash：Order code sequence AA-8A or AA-E4. LS2051allows to erase whole 2KB Flash space. When SPI receives these two bytes successively, chips enter sequence of Flash erasing operation: the first byte is AA, and the second byte is 8A or E4.
- LS2051’s SPI does not support read operation to Flash in order to strength lock performance.

Erasing Sequence of Flash

When SPI receives the order code AA and 8A, chips enter the flash erasing sequence, the sequence is shown in Figure 7. The order code AA and 8A must be transmitted successively , any other information should not be inserted. Other order codes should not sent to SPI interface during the erasing period, otherwise it will cause failure of erasing operation.

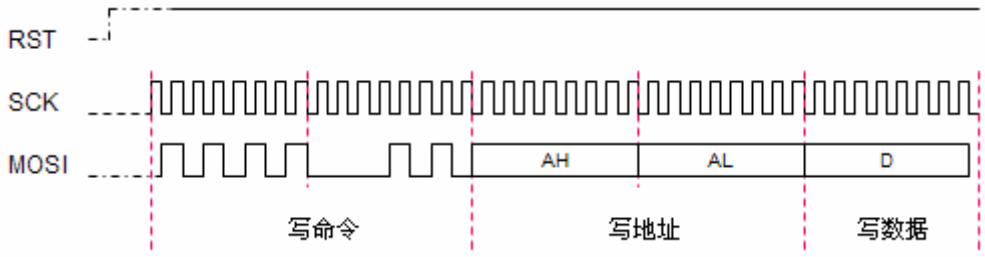


Erasing order Erasing

Figure 7: Erasing Sequence of Flash

Writing sequence of Flash

When SPI receives AA-50-AX-AY-data bytes, chips enter the flash writing sequence. Order codes should be transmitted successively, any other information should not be inserted. The flash writing sequence is shown in Figure 8. During the writing period, the other order codes should not be transmitted to SPI interface, or it will cause failure of writing operation. There will be auto-verification after finishing writing a byte. If it is correct, MOSI will output 0, if not, MOSI will output 1. The failure of some byte's writing will not affect the writing operation of following bytes, but rewriting of the byte will not succeed.



Write order write address write data

Figure 8: Flash Sequence

Limit Parameter

The LS2051's limit parameters is as Table 4.

Operating temperature	-40℃ to +85℃
Storage temperature	-60℃ to +125℃
Voltage on Pin with Respect to Ground	-1.0V to +7.0V
V_{CC}	6.6V
I_{CC}	25.0mA

Table 4: LS2051 Limit Parameters

Current characteristic

LS2051's direct current characteristic is as Table 5.

Sym	Parameter	Condition	Min	Max	Units
V_{IL}	Input Low-voltage		-0.5	0.2 Vcc - 0.1	V
V_{IH}	Input High-voltage	(Except XTAL1, RST)	0.2 Vcc + 0.9	Vcc + 0.5	V
VIH1	Input High-voltage	(XTAL1, RST)	0.7 Vcc	Vcc + 0.5	V
V_{OL}	Output Low-voltage	Iol = 20 mA, Vcc = 5V		0.5	V
V_{OH}	Output High-voltage	Ioh = -80 μA, Vcc = 5V*10%	2.4		V
I_{IL}	Logical 0 Input Current	Vin = 0.45V		-50	μA
I_{TL}	Logical 1 to 0 Transition Current	Vin = 2V, Vcc = 5V*10%		-750	μA
I_{LI}	Input Leakage Current	0 < Vin < Vcc		10	μA
V_{OS}	Comparator Input Offset Voltage	Vcc= 5V		20	mV
V_{CM}	Comparator Input Common Mode Voltage		0	Vcc	V
RRST	Reset Pull-down Resistor		30	70	KΩ
C_{IO}	Pin Capacitance	Test Freq. = 1 MHz, TA = 25°C		10	pF
I_{CC}	Power Supply Current	Active Mode, 12 MHz, Vcc=5V		20	mA

Table 5: Current Parameters of LS2051

Notes ：

Under the stable circumstance (not instant), IOL is restricted the following

- The max allowable value of each port I_{OL} is 20mA.
- The max sum of all the output port I_{OL} is 80mA.
- If I_{OL} exceeds test condition allowable value, and then V_{OL} may exceed allowable value by relatively, but it can guarantee port value will exceed value gained from test condition as in Table 5.

Instruction set

Mnemonics definition table

In order to convenience the description to LS2051, the mnemonics are as table 6.

Mnemonics	Description
Rn	Register R7±R0 of the currently selected Register Bank.
Direct	8-bit internal data location's address. This could be an Internal Data RAM location (0±127) or a SFR [i.e., I/O port, control register, status register, etc. (128±255)].
@Ri	8-bit internal data RAM location (0±255) addressed indirectly through register R1 or R0.
#data	8-bit constant included in instruction.
#data16	16-bit constant included in instruction.
Addr16	16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.
Addr11	11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction
Rel	Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is b128 to a127 bytes relative to first byte of the following instruction.
bit	Direct Addressed bit in Internal Data RAM or Special Function Register.

Table 6: Mnemonics Descriptions

Instructions

LS2051 instructions and their operations are show in Table 7.

Instruction	Byte	Oscillator Period	Description
ARITHMETIC OPERATIONS			
1 ADD A,Rn	1	12	Add register to Accumulator
2 ADD A,direct	2	12	Add direct byte to Accumulator
3 ADD A,@Ri	1	12	Add indirect RAM to Accumulator
4 ADD A,#data	2	12	Add immediate data to Accumulator

5 ADDC A,Rn	1	12	Add register to Accumulator with Carry
6 ADDC A,direct	2	12	Add direct byte to Accumulator with Carry
7 ADDC A,@Ri	1	12	Add indirect RAM to Accumulator with Carry
8 ADDC A,#data	2	12	Add immediate data to Acc with Carry
9 SUBB A,Rn	1	12	Subtract Register from Acc with borrow
10 SUBB A,direct	2	12	Subtract direct byte from Acc with borrow
11 SUBB A,@Ri	1	12	Subtract indirect RAM from ACC with borrow
12 SUBB A，#data	2	12	Subtract immediate data from Acc with borrow
13 INC A	1	12	Increment Accumulator
14 INC Rn	1	12	Increment register
15 INC direct	2	12	Increment direct byte
16 INC @Ri	1	12	Increment direct RAM
17 INC DPTR	1	12	Increment Data Pointer
18 DEC A	1	12	Decrement Accumulator
19 DEC Rn	1	12	Decrement Register
20 DEC direct	2	12	Decrement direct byte
21 DEC @Ri	1	12	Decrement indirect RAM
22 MUL AB	1	12	Multiply A & B
23 DIV AB	1	12	Divide A by B
24 DA A	1	12	Decimal Adjust Accumulator
LOGICAL OPERATIONS			
25 ANL A,Rn	1	12	AND Register to Accumulator
26 ANL A,direct	2	12	AND direct byte to Accumulator
27 ANL A,@Ri	1	12	AND indirect RAM to Accumulator
28 ANL A,#data	2	12	AND immediate data to Accumulator
29 ANL direct,A	2	12	AND Accumulator to direct byte
30 ANL direct,#data	3	12	AND immediate data to direct byte
31 ORL A,Rn	1	12	OR register to Accumulator
32 ORL A,direct	2	12	OR direct byte to Accumulator
33 ORL A,@Ri	1	12	OR indirect RAM to Accumulator
34 ORL A,#data	2	12	OR immediate data to Accumulator
35 ORL direct,A	2	12	OR Accumulator to direct byte
36 ORL direct,#data	3	12	OR immediate data to direct byte
37 XRL A,Rn	1	12	Exclusive-OR Register to Accumulator
38 XRL A,direct	2	12	Exclusive-OR direct byte to Accumulator
39 XRL A,@Ri	1	12	Exclusive-OR indirect RAM to Accumulator
40 XRL A,#data	2	12	Exclusive-OR immediate data to Accumulator
41 XRL direct,A	2	12	Exclusive-OR Accumulator to direct byte

42 XRL direct,#data	3	12	Exclusive-OR immediate data to direct byte
43 CLR A	1	12	Clear Accumulator
44 CPL A	1	12	Complement Accumulator
45 RL A	1	12	Rotate Accumulator Left
46 RLC A	1	12	Rotate Accumulator Left through the Carry
47 RR A	1	12	Rotate Accumulator Right
48 RRC A	1	12	Rotate Accumulator Right through the Carry
49 SWAP A	1	12	Swap nibbles within the Accumulator
DATA TRANSFER			
50 MOV A,Rn	1	12	Move register to Accumulator
51 MOV A,direct	2	12	Move direct byte to Accumulator
52 MOV A,@Ri	1	12	Move indirect RAM to Accumulator
53 MOV A,#data	2	12	Move immediate data to Accumulator
54 MOV Rn，A	1	12	Move Accumulator to register
55 MOV Rn,direct	2	12	Move Accumulator to register
56 MOV Rn,gdata	2	12	Move Accumulator to register
57 MOV direct,A	2	12	Move Accumulator to direct byte
58 MOV direct,Rn	2	12	Move register to direct byte
59 MOV direct1, direct2	3	12	Move direct2 byte to direct1
60 MOV direct,@Ri	2	12	Move indirect RAM to direct byte
61 MOV direct,#data	3	12	Move immediate data to direct byte
62 MOV @Ri,A	1	12	Move Accumulator to indirect RAM
63 MOV @Ri,direct	2	12	Move direct byte to indirect RAM
64 MOV @Ri,#data	2	12	Move direct byte to indirect RAM
65 MOV DPTR, #data16	3	12	Load Data Pointer with a 16-bit constant
66 MOVC A, @A+DPTR	1	24	Move Code byte relative to DPTR to Acc
67 MOVC A, @A+PC	1	24	Move Code byte relative to PC to Acc
68 PUSH direct	2	12	Push direct byte onto stack
69 POP direct	2	12	Pop direct byte from stack
70 XCH A,Rn	1	12	Exchange register with Accumulator
71 XCH A,direct	2	12	Exchange direct byte with Accumulator
72 XCH A,@Ri	1	12	Exchange indirect RAM with Accumulator
73 XCHD A,@Ri	1	12	Exchange low-order Digit indirect RAM with Acc
BOOLEAN VARIABLE MANIPULATION			
74 CLR C	1	12	Clear Carry
75 CLR bit	2	12	Clear direct bit
76 SETB C	1	12	Set Carry

77 SETB bit	2	12	Set direct bit
78 CPL C	1	12	Complement Carry
79 CPL bit	2	12	Complement direct bit
80 ANL C,bit	2	12	AND direct bit to CARRY
81 ANL C,/bit	2	12	AND complement of direct bit to Carry
82 ORL C,bit	2	12	OR direct bit to CARRY
83 ORL C,/bit	2	12	OR complement of direct bit to Carry
84 MOV C,bit	2	12	Move direct bit to Carry
85 MOV bit,C	2	12	Move Carry to direct bit
86 JC rel	2	12	Jump if Carry is set
87 JNC rel	2	12	Jump if Carry is not set
88 JB bit,rel	3	12	Jump if direct Bit is set
89 JNB bit,rel	3	12	Jump if direct Bit is not set
90 JBC bit,rel	3	12	Jump if direct Bit is set & clear bit
PROGRAM BRANCHING			
91 ACALL addr11	2	12	Absolute Subroutine Call
92 LCALL addr16	3	12	Long Subroutine Call
93 RET	1	12	Return from Subroutine
94 RETI	1	12	Return from interrupt (Program 1). Stop running (Program 2)
95 AJMP addr11	2	12	Absolute Jump (2K)
96 LJMP addr16	3	12	Long Jump (64K)
97 SJMP rel	2	12	Short Jump (relative addr, -128～+127bytes)
98 JMP @A+DPTR	1	12	Jump indirect relative to the DPTR
99 JZ rel	2	12	Jump if Accumulator is Zero
100 JNZ rel	2	12	Jump if Accumulator is not Zero
101 CJNE A,direct,rel	3	12	Compare direct byte to Acc and Jump if Not Equal
102 CJNE A,#data,rel	3	12	Compare immediate to Acc and Jump if Not Equal
103 CJNE Rn, #data, rel	3	12	Compare immediate to register and Jump if Not Equal
104 CJNE @Ri, #data, rel	3	12	Compare immediate to indirect and Jump if Not Equal
105 DJNZ Rn,rel	2	12	Decrement register and Jump if Not Zero
106 DJNZ direct,rel	3	12	Decrement direct byte and Jump if Not Zero
107 NOP	1	12	No Operation
INSTRUCTIONS REALETED TO CONCURRENT PROGRAMS			
MOV 0FEH,#data	3	12	Program 2 Stops Running
MOV 0FCH,#addr8	3	12	Jump and Set the Synchronizing-bit to be One if the Synchronizing-bit is Zero

MOV 0FBH,#addr8	3	12	Jump and Set the Synchronizing-bit to be Zero if the Synchronizing-bit is not Zero
MOV 0FFH,#addr8	3	12	Load Program 2 to execute (Start Address is addr8)
MOV 0FDH, #1	3	12	Set the Synchronizing-bit to be One
MOV 0FDH, #0	3	12	Set the Synchronizing-bit to be Zero

Table 7: LS2051Instruction Set

Examples for Two programs to Execute Concurrently

1. The sample program for running two programs concurrently: two programs execute their own algorithms to calculate something concurrently, and then the results will be added up to produce the final results. The final result will be then output through P1 port. The final results are 07H, 38H, 39H, 07H, 07H, 04H, 0BH. The program is as follows.

```
MAIN:                ;  Major program executes in the first path
MOV 0FFH,#ROAD1      ;  Turn program in the first path on
MOV A,#2             ; A=2
ADD A,#5             ; A=7
INC A                ; A=8
SUBB A,#4            ; A=4
MOV R0,A
MOV 0FBH,#$          ;  Judge whether synchronous flag is 1, if it is 1, then continue
;  to execute; if it is 0, then execute this statement always.
MOV A,12H
MOV P1,A             ; A=7
MOV A,R0
MOV P1,A             ; Output from P1 port, A=4
ADD A,12H            ; A=0BH
MOV P1,A             ;  Output from P1 port
AJMP  $              ;  Program in the first path cycles here

ROAD1:               ; The entrance address of program in the second path
MOV A,#7             ; A=7

MOV P1,A             ; Output from P1 port
MOV B,#8
```

```
MUL AB              ；B=0 A=56
MOV P1,A
INC A               ；A=57
MOV P1,A
SUBB A,#50          ；A=7
MOV P1,A
MOV 12H,A
MOV 0FDH,#1         ；Set synchronous flag to be 1
RETI                ；The second program ends. This can be replaced by MOV 0FEH
;  # instant number.
```

2. Play two songs by two programs concurrently: the main program play the song (named Sweet Olive Blooming in August) edited by C language, and the output port is P1.0. The second program play the song named Happy Birthday, assembled by assembler, and output through Port P1.1. The example includes the following programs: music_c_asm.c, road1.a51, loadp.a51.

Sweet Olive Blooming in August: music_c_asm.c:

```
extern void loadp(void)；
#include <reg2051.h>
#include <intrins.h>
// crystal applied in this example is 11.0592MHZ
unsigned char n=0;          //n is beat constant variable
unsigned char temp_th1=0,temp_tl1=0;
unsigned char code music_tab[] =
{       //format is：frequency constant, beat constant, frequency constant, beat constant.
0x18, 0x30, 0x1C , 0x10,   0x20, 0x40, 0x1C , 0x10,
0x18, 0x10, 0x20 , 0x10,   0x1C, 0x10, 0x18 , 0x40,
0x1C, 0x20, 0x20 , 0x20,   0x1C, 0x20, 0x18 , 0x20,
0x20, 0x80, 0xFF , 0x20,   0x30, 0x1C, 0x10 , 0x18,
0x20, 0x15, 0x20 , 0x1C,   0x20, 0x20, 0x20 , 0x26,
0x40, 0x20, 0x20 , 0x2B,   0x20, 0x26, 0x20 , 0x20,
0x20, 0x30, 0x80 , 0xFF,   0x20, 0x20, 0x1C , 0x10,

0x18, 0x10, 0x20 , 0x20,   0x26, 0x20, 0x2B , 0x20,
0x30, 0x20, 0x2B , 0x40,   0x20, 0x20, 0x1C , 0x10,
0x18, 0x10, 0x20 , 0x20,   0x26, 0x20, 0x2B , 0x20,
```

```
0x30, 0x20, 0x2B , 0x40,   0x20, 0x30, 0x1C , 0x10,
0x18, 0x20, 0x15 , 0x20,   0x1C, 0x20, 0x20 , 0x20,
0x26, 0x40, 0x20 , 0x20,   0x2B, 0x20, 0x26 , 0x20,
0x20, 0x20, 0x30 , 0x80,   0x20, 0x30, 0x1C , 0x10,
0x20, 0x10, 0x1C , 0x10,   0x20, 0x20, 0x26 , 0x20,
0x2B, 0x20, 0x30 , 0x20,   0x2B, 0x40, 0x20 , 0x15,
0x1F, 0x05, 0x20 , 0x10,   0x1C, 0x10, 0x20 , 0x20,
0x26, 0x20, 0x2B , 0x20,   0x30, 0x20, 0x2B , 0x40,
0x20, 0x30, 0x1C , 0x10,   0x18, 0x20, 0x15 , 0x20,
0x1C, 0x20, 0x20 , 0x20,   0x26, 0x40, 0x20 , 0x20,
0x2B, 0x20, 0x26 , 0x20,   0x20, 0x20, 0x30 , 0x30,
0x20, 0x30, 0x1C , 0x10,   0x18, 0x40, 0x1C , 0x20,
0x20, 0x20, 0x26 , 0x40,   0x13, 0x60, 0x18 , 0x20,
0x15, 0x40, 0x13 , 0x40,   0x18, 0x80, 0x00
};
void int0() interrupt 1   // apply interrupt 0 to controlling beat
{
TH0=0xd8;
TL0=0xef;
n--;
}
void int1() interrupt 3   // apply interrupt 1 to controlling music in the second path
{
TL1=temp_tl1;
TH1=temp_th1;
P1_1=~P1_1;
}
void delay (unsigned char m)   // control frequency delay
{
unsigned i=3*m;
while(--i);
}

void delayms(unsigned char a)   // millisecond delay sub-programs
{
while(--a);                    //adopt (--a) rather than while(a--);
}
```

```
void main()
{
unsigned char p,m;                  //m is frequency constant variable
unsigned char i=0;
TMOD&=0xff;
TMOD|=0x11;
TH0=0xd8;
TL0=0xef;
IE=0x8a;
loadp();                            //load the second program
play:
while(1)
{
a:
p=music_tab[i];
if(p==0x00)
{
i=0;
delayms(1000);
goto play;
}     // if there is end signal, delay 1 second, do it again from the beginning.
else if(p==0xff)
{
i=i+1;
delayms(100);
TR0=0;
goto a;
}   // If there is rest, delay100ms, and then continue to select the next note.
else

{
m=music_tab[i++];
n=music_tab[i++];
}           // Select frequency constants and beat constants
TR0=1;    //turn on timer 0
while(n!=0)
{
```

```
P1_0=~P1_0;
delay(m);
}          // wait for completion of beat and output audio frequency from P1 port.
TR0=0;    // turn off timer 0
}
}
```

Loader program of the second program loadp.a51

```
extrn code(road1)
NAME       LOADP
LOADP1    SEGMENT    CODE
PUBLIC     loadp
RSEG        LOADP1
loadp:
USING       0
MOV         0FFH,#ROAD1              ;start the second path and execute ROAD1 on it.
RET
END
```

Happy Birthday road1.a51

```
extrn data(temp_th1)
extrn data(temp_tl1)

NAME          ROAD1
ROAD11       SEGMENT    CODE
PUBLIC   ROAD1
RSEG          ROAD11
ROAD1:

USING          0
start0:
mov 60h,#00h                          ;select indicator of sight-sing code
next: mov a,60h                        ; load A to sight-sing code
mov dptr,#table                       ;till table select sight-sing code
movc a,@a+dptr
mov  r2,a                                    ; store selected sight-sing codes to R2
temporarily
jz end1                                 ;Whether select 00( end code)?
```

```
anl a,#0fh                    ; if not, select low 4 bit( note code)
mov r5,a                      ; store beat code to R5
mov a,r2                      ; A load A to sight-sing code again
swap a                        ; interchange of high and low four bits
anl a,#0fh                    ; select low four bits( notes)
jnz sing                      ; whether the selected note is zero?
clr tr1                       ; If so, it is aphonic
jmp d1
sing: dec a                   ; selected note minus 1( not include 0)
mov 63h,a                     ;store (22H).
rl a                          ;multiply 2
mov dptr,#table1              ;till table1 select comparable high byte count
;value
movc a,@a+dptr
mov th1,a                     ; store selected byte to TH1
mov temp_th1,a                ;store selected byte to (21H)
mov a,63h                     ;reload selected note
rl a                          ;multiply 2
inc a                         ;plus 1
movc a,@a+dptr                ;till table1 select comparable low byte count
; value
mov tl1,a                     ;store the selected high-bit type to TL1
mov temp_tl1,a                ;store the selected high-bit byte to (20H)
setb tr1                      ;start timer 0
d1: call delay                ; basic unit time 1/4 beat 187millisecond
inc 60h                       ;select sight-singing indicator plus one

jmp next                      ; select a code
end1: clr tr1                 ; terminate timer1
jmp start0                    ;loop?
delay: mov r7,#02h            ;187 millisecond
d2: mov r4,#187
d3: mov r3,#248
djnz r3,$
djnz r4,d3
djnz r7,d2
djnz r5,delay                 ;decide beat
```

```
ret
table1:
dw 64260,64400,64524,64580
dw 64684,64777,64820,64898
dw 64968,65030,65058,65110
dw 65157,65178,65217
table:
;1
db 82h,01h,81h,94h,84h,0b4h,0a4h,04h,82h,01h,81h,94h,84h,0c4h,0b4h,04h
;2
db 82h,01h,81h,0f4h,0d4h,0b4h,0a4h,94h,0e2h,01h,0e1h,0d4h,0b4h,0c4h,0b4h,04h
;3
db 82h,01h,81h,94h,84h,0b4h,0a4h,04h,82h,01h,81h,94h,84h,0c4h,0b4h,04h
;4
db 82h,01h,81h,0f4h,0d4h,0b4h,0a4h,94h,0e2h,01h,0e1h,0d4h,0b4h,0c4h,0b4h,04h,00h
RET
END
```

Product order

Frequency	voltage	Product Order	packaging	Operating range
0 to 24MHz	3.0V to 5.5V	LS2051-224SJI LS4051-224SJI	20S	-40℃to 85℃
0 to 24MHz	3.0V to 5.5V	LS2051-224PJI LS4051-224PJI	20P3	-40℃to 85℃

Packaging information

LS2051/LS4051 adopts 20 leads and 0.300" wide SOIC packing, as in Figure 9, the dimensions are in inches and millimeters.

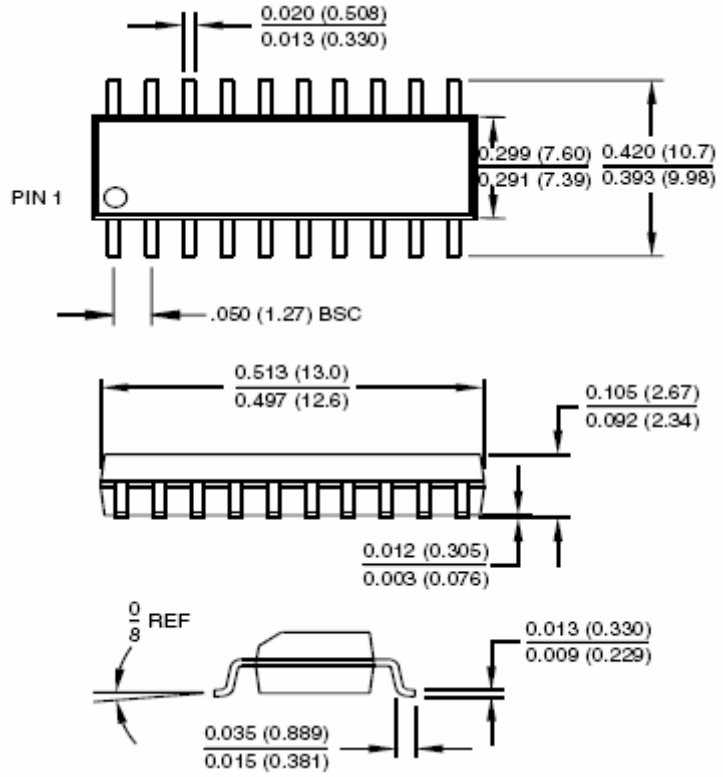


Figure 9 LS2051/LS4051 packaging（in SOIC）

LS2051/LS4051 adopts 20 leads and 0.300” wide PDIP packing, as in Figure 10, the dimensions are in inches and millimeters.

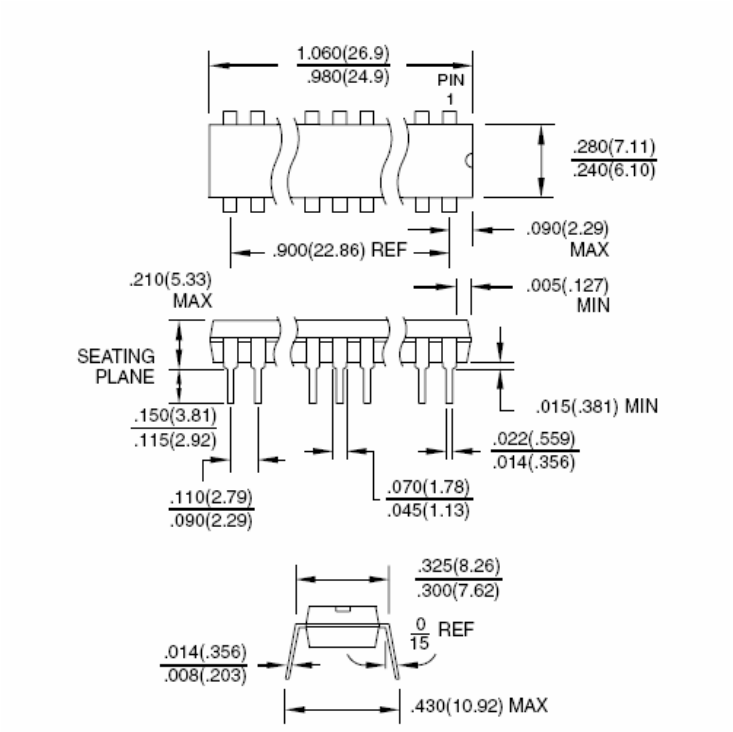


Figure 10 LS2051/LS4051 packaging（in PDIP）