BAB 6
KESIMPULAN DAN SARAN

6.1. Kesimpulan

Berdasarkan hasil penelitian yang telah dilakukan, maka kesimpulan dari penelitian ini adalah dengan implementasi teknologi mikrotontroller AT89S51 dihasilkan model kunci elektronik yang dapat membatasi akses pengguna ke dalam ruangan. Kunci elektronik yang dihasilkan memiliki tingkat keamanan, kemudahan dan kenyaman yang lebih baik dari kunci konvensional. Kunci elektronik juga dilengkapi dengan display LCD dan sirine.

6.2. Saran

Dari penelitian yang telah dilakukan, ada beberapa saran yang dapat diberikan penulis untuk lebih menyempurnakan hasil penelitian ini di masa mendatang. Adapun saran penulis sebagai berikut:

1. Sumber tegangan listrik kunci elektronik dapat menggunakan baterai basah agar tidak mudah di-reset.

2. Keamanan kunci elektronik dapat ditingkatkan dengan menambah jumlah kode pin.
DAFTAR PUSTAKA

Budiharto, W., 2005, Perancangan Sistem dan Aplikasi Mikrokontroler, PT. Elex Media Komputindo, Jakarta.

Budioko, T., 2005, Belajar dengan mudah dan cepat Pemograman Bahasa C dengan SDCC (Small Device C Compiler) pada Mikrokontroler AT89X051/AT89C51/52 Teori, Simulasi dan Aplikasi, ed. 1, cet. 1, Gava Media, Yogyakarta.


LAMPIRAN
Lampiran 1:

Data Sheet Mikrokontroler
AT89S51
features
Compatible with MC68S1 Products
4K Bytes of In-System Programmable (ISP) Flash Memory
- Endurance: 1000 Write/Erase Cycles
- 4.0V to 5.5V Operating Range
- Fully Static Operation: 0 Hz to 33 MHz
- Time-level Program Memory Lock
- 128 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Two 16-bit Timer/Counters
- Six Interrupt Sources
- Full Duplex UART Serial Channel
- Low-power Idle and Power-down Modes
- Interrupt Recovery from Power-down Mode
- Watchdog Timer
- Dual Data Pointer
- Power-off Flag
- Fast Programming Time
- Flexible ISP Programming (Byte and Page Mode)
- Green (PlatHledge-free) Packaging Option

Description
The AT89S51 is a low-power, high-performance CMOS 8-bit microcontroller with 4K bytes of In-System Programmable Flash memory. The device is manufactured using Atmel's high-density nonvolatile memory technology and is compatible with the industry-standard 8051 instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with In-System Programmable Flash on the same chip, the Atmel AT89S51 is a powerful microcontroller which provides a highly flexible and cost-effective solution to many embedded control applications.

The AT89S51 provides the following standard features: 4K bytes of Flash, 128 bytes of RAM, 32 I/O lines, Watchdog timer, two data pointers, two 16-bit timer/counters, a vectored two-level interrupt architecture, a full duplex serial port, on-chip oscillator, and clock circuitry. In addition, the AT89S51 is designed with static logic for operation over zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timers/counters, serial port, and interrupt system to continue functioning. The Power-down mode saves the RAM contents but freezes the oscillator, disabling all other chip functions until the next external interrupt or hardware reset.
Pin Description

1. VCC
   Supply voltage (all packages except 40-PDIP).

2. GND
   Ground (all packages except 42-PDIP; for 42-PDIP GND connects only the logic core and the embedded program memory).

3. VDD
   Supply voltage for the 40-PDIP which connects only the logic core and the embedded program memory.

4. PWRVDD
   Supply voltage for the 40-PDIP which connects only the I/O Pad Drivers. The application board MUST connect both VDD and PWRVDD to the board supply voltage.

5. PWRGND
   Ground for the 40-PDIP which connects only the I/O Pad Drivers. PWRGND and GND are weakly connected through the common silicon substrate, but not through any metallic link. The application board MUST connect both GND and PWRGND to the board ground.

6. Port 0
   Port 0 is an 8-bit open-drain bi-directional I/O port. As an output port, each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs.
   Port 0 can also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode, P0 has internal pull-ups.
   Port 0 also receives the code bytes during Flash programming and outputs the code bytes during program verification. External pull-ups are required during program verification.

7. Port 1
   Port 1 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins, they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (Ih) because of the internal pull-ups.
   Port 1 also receives the low-order address bytes during Flash programming and verification.

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>Alternate Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.5</td>
<td>MOSI (used for In-System Programming)</td>
</tr>
<tr>
<td>P1.6</td>
<td>MISO (used for In-System Programming)</td>
</tr>
<tr>
<td>P1.7</td>
<td>SCK (used for In-System Programming)</td>
</tr>
</tbody>
</table>
Port 2

Port 2 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 2 output buffers can sink/source four TTL inputs. When 1's are written to Port 2 pins, they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (Ih) because of the internal pull-ups.

Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @ D'PTR). In this application, Port 2 uses strong internal pull-up when emitting 1's. During accesses to external data memory that use 8-bit addresses (MOVX @ Rl), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

Port 3

Port 3 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 3 output buffers can sink/source four TTL inputs. When 1's are written to Port 3 pins, they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (Ih) because of the pull-ups.

Port 3 receives some control signals for Flash programming and verification.

Port 3 also serves the functions of various special features of the AT89S51, as shown in the following table.

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>Alternate Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3.0</td>
<td>RXD (serial input port)</td>
</tr>
<tr>
<td>P3.1</td>
<td>TXD (serial output port)</td>
</tr>
<tr>
<td>P3.2</td>
<td>INT0 (external interrupt 0)</td>
</tr>
<tr>
<td>P3.3</td>
<td>INT1 (external interrupt 1)</td>
</tr>
<tr>
<td>P3.4</td>
<td>T0 (timer 0 external input)</td>
</tr>
<tr>
<td>P3.5</td>
<td>T1 (timer 1 external input)</td>
</tr>
<tr>
<td>P3.6</td>
<td>WR (external data memory write strobe)</td>
</tr>
<tr>
<td>P3.7</td>
<td>RD (external data memory read strobe)</td>
</tr>
</tbody>
</table>

RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device. This pin drives high for 98 oscillator periods after the Watchdog times out. The DISRT0 bit in SFR A20H (address 8EH) can be used to disable this feature. In the default state of bit DISRT0, the RESET RST pin out feature is enabled.

ALE/PROG

Address Latch Enable (ALE) is an output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input (PROG) during Flash programming.
In normal operation, ALE is emitted at a constant rate of 1/6 the oscillator frequency and may be used for external timing or clocking purposes. Note however, that one ALE pulse is skipped during each access to external data memory.

If desired, ALE operation can be disabled by setting bit 0 of 5Fh location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the ALE-disable bit has no effect if the microcontroller is in external execution mode.

**PSEN**

Program Store Enable (PSEN) is the read strobe to external program memory.

While the AT89S51 is executing code from external program memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to external data memory.

**EA/VPP**

External Access Enable. EA must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, EA will be internally latched on reset. EA should be strapped to VDD for internal program executions.

This pin also receives the 12-volt programming enable voltage (Vpp) during Flash programming.

**XTAL1**

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

**XTAL2**

Output from the inverting oscillator amplifier.

**Special Function Registers**

A map of the on-chip memory area called the Special Function Register (SFR) space is shown in Table 5-1.

Note that not all of the addresses are occupied, and unoccupied addresses may not be implemented on the chip. Read accesses to these addresses will in general return random data, and write accesses will have an indeterminate effect.

**AT89S51**
### AT89S51 SFR Map and Reset Values

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H</td>
<td>ACC</td>
</tr>
<tr>
<td>01H</td>
<td>PSW</td>
</tr>
<tr>
<td>02H</td>
<td>IP</td>
</tr>
<tr>
<td>03H</td>
<td>P3</td>
</tr>
<tr>
<td>04H</td>
<td>IE</td>
</tr>
<tr>
<td>05H</td>
<td>P2</td>
</tr>
<tr>
<td>06H</td>
<td>SCON</td>
</tr>
<tr>
<td>07H</td>
<td>TH1</td>
</tr>
<tr>
<td>08H</td>
<td>TH0</td>
</tr>
<tr>
<td>09H</td>
<td>TCON</td>
</tr>
<tr>
<td>0AH</td>
<td>TMOD</td>
</tr>
<tr>
<td>0BH</td>
<td>OBUF</td>
</tr>
<tr>
<td>0CH</td>
<td>PCON</td>
</tr>
</tbody>
</table>

*Note: software should not write 1s to these unlisted locations, since they may be used in future products to invoke new features. In that case, the reset or inactive values of the new bits will always be 0.*

**Interrupt Registers:** The individual interrupt enable bits are in the IE register. Two priorities can be set for each of the five interrupt sources in the IP register.
### Table 5.2. AUXR: Auxiliary Register

<table>
<thead>
<tr>
<th></th>
<th>Address = 8EH</th>
<th>Reset Value = XXX00000B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not Bit Addressable</td>
<td></td>
</tr>
<tr>
<td>Bit</td>
<td></td>
<td>WIDLE</td>
</tr>
<tr>
<td>-----</td>
<td>---</td>
<td>------</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **WIDLE**: Disable/Enable ALE
- **DISRTO**: Disable/Enable Reset-out
- **DISALE**: Disable/Enable Reset-out
- **ALE**: Active only during a MOVX or MOVY instruction
- **ALE**: Emitted at a constant rate of 1/6 the oscillator frequency

**Dual Data Pointer Registers**: To facilitate accessing both internal and external data memory, two banks of 16-bit Data Pointer Registers are provided: DP0 at SFR address locations 82H-83H and DP1 at 84H-85H. Bit DPS = 0 in SFR AUXR1 selects DP0 and DPS = 1 selects DP1. The user should ALWAYS initialize the DPS bit to the appropriate value before accessing the respective Data Pointer Register.

**Power Off Flag**: The Power Off Flag (POF) is located at bit 4 (PCCN.4) in the PCCN SFR. POF is set to "1" during power up. It can be set and reset under software control and is not affected by reset.
Table 5-3. AUXR1: Auxiliary Register 1

<table>
<thead>
<tr>
<th>AUXR1 Address</th>
<th>Reset Value = XXXXXXX0S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Not Bit Addressable</td>
<td></td>
</tr>
<tr>
<td>Reserved for future expansion</td>
<td></td>
</tr>
<tr>
<td>DPH</td>
<td>Data Pointer Register Select</td>
</tr>
<tr>
<td>DPS</td>
<td>Selects DPR Registers DPOL, DPOLH</td>
</tr>
<tr>
<td>0</td>
<td>Selects DPR Registers DP0L, DP0H</td>
</tr>
</tbody>
</table>

Memory Organization

MCS-51 devices have a separate address space for Program and Data Memory. Up to 54K bytes each of external Program and Data Memory can be addressed.

1 Program Memory

If the EA pin is connected to GND, all program fetches are directed to external memory.

On the AT89S51, if EA is connected to VCC, program fetches to addresses 0000H through FFFFH are directed to internal memory and fetches to addresses 1000H through FFFFH are directed to external memory.

2 Data Memory

The AT89S51 implements 128 bytes of on-chip RAM. The 128 bytes are accessible via direct and indirect addressing modes. Stack operations are examples of indirect addressing, so the 128 bytes of data RAM are available as stack space.

Watchdog Timer (One-time Enabled with Reset-out)

The WDT is intended as a recovery method in situations where the CPU may be subjected to software upsets. The WDT consists of a 14-bit counter and the Watchdog Timer Reset (WDTRST) SFR. The WDT is disabled to prevent a system reset. To enable the WDT, a user must write 01EH and 01EH in sequence to the WDTST register (SFR location 0A9H). When the WDT is enabled, it will increment every machine cycle while the oscillator is running. The WDT timeout period is dependent on the external clock frequency. There is no way to disable the WDT except through reset (either hardware reset or WDT overflow reset). When WDT overflows, it will drive an output RESET HIGH pulse at the RST pin.

1 Using the WDT

To enable the WDT, a user must write 01EH and 01EH in sequence to the WDTRST register (SFR location 0A9H). When the WDT is enabled, the user needs to service it by writing 01EH and 01EH to WDTRST to avoid a WDT overflow. The 14-bit counter overflows when it reaches 16383 (FFFFH), and this will reset the device. When the WDT is enabled, it will increment every machine cycle while the oscillator is running. This means the user must reset the WDT at least
every 16383 machine cycles. To reset the WDT the user must write 01EH and 0EH to WDTST. WDTST is a write-only register. The WDT counter cannot be read or written. When WDT overflows, it will generate an output RESET pulse at the RST pin. The RESET pulse duration is 98xTOS2c, where TOSC = 1/FOSC. To make the best use of the WDT, it should be serviced in those sections of code that will periodically be executed within the time required to prevent a WDT reset.

2 WDT During Power-down and Idle
In Power-down mode the oscillator stops, which means the WDT also stops. While in Power-down mode, the user does not need to service the WDT. There are two methods of exiting Power-down mode: by a hardware reset or via a level-activated external interrupt, which is enabled prior to entering Power-down mode. When Power-down is exited with hardware reset, servicing the WDT should occur as it normally does whenever the AT89S51 is reset. Exiting Power-down with an interrupt is significantly different. The interrupt is held low long enough for the oscillator to stabilize. When the interrupt is brought high, the interrupt is servied. To prevent the WDT from resetting the device while the interrupt pin is held low, the WDT is not started until the interrupt is pulled high. It is suggested that the WDT be reset during the interrupt service for the interrupt used to exit Power-down mode.

To ensure that the WDT does not overflow within a few states of exiting Power-down, it is best to reset the WDT just before entering Power-down mode.

Before going into the IDLE mode, the WDIDLE bit in SPR AUXR is used to determine whether the WDT continues to count if enabled. The WDT keeps counting during IDLE (WDIDLE bit = 0) as the default state. To prevent the WDT from resetting the AT89S51 while in IDLE mode, the user should always set up a timer that will periodically exit IDLE, service the WDT, and reenter IDLE mode.

With WDIDLE bit enabled, the WDT will stop to count in IDLE mode and resumes the count upon exit from IDLE.

3 UART
The UART in the AT89S51 operates the same way as the UART in the AT89C51. For further information on the UART operation, please click on the document link below:

http://www.atmel.com/dyn/resources/prod_documents/DOC4316.PDF

4 Timer 0 and 1
Timer 0 and Timer 1 in the AT89S51 operate the same way as Timer 0 and Timer 1 in the AT89C51. For further information on the timers' operation, please click on the document link below:

http://www.atmel.com/dyn/resources/prod_documents/DOC4316.PDF

AT89S51
0. Interrupts

The AT89S51 has a total of five interrupt vectors: two external interrupts (INT0 and INT1), two timer interrupts (Timers 0 and 1), and the serial port interrupt. These interrupts are all shown in Figure 10-1.

Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE. IE also contains a global disable bit, EA, which disables all interrupts at once.

Note that Table 10-1 shows that bit positions IE.6 and IE.5 are unimplemented. User software should not write 1s to these bit positions, since they may be used in future AT89 products.

The Timer 0 and Timer 1 flags, T0IF and T1IF, are set at SSP2 of the cycle in which the timers overflow. The values are then polled by the circuitry in the next cycle.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Position</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA</td>
<td>IE.7</td>
<td>Enables all interrupts. If EA = 0, no interrupt is acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.</td>
</tr>
<tr>
<td>o</td>
<td>IE.6</td>
<td>Reserved</td>
</tr>
<tr>
<td>o</td>
<td>IE.5</td>
<td>Reserved</td>
</tr>
<tr>
<td>ES</td>
<td>IE.4</td>
<td>Serial Port interrupt enable bit</td>
</tr>
<tr>
<td>ET1</td>
<td>IE.3</td>
<td>Timer 1 interrupt enable bit</td>
</tr>
<tr>
<td>EX1</td>
<td>IE.2</td>
<td>External Interrupt 1 enable bit</td>
</tr>
<tr>
<td>ETO</td>
<td>IE.1</td>
<td>Timer 0 interrupt enable bit</td>
</tr>
<tr>
<td>EX0</td>
<td>IE.0</td>
<td>External Interrupt 0 enable bit</td>
</tr>
</tbody>
</table>

User software should never write 1s to reserved bits, because they may be used in future AT89 products.
1. Oscillator Characteristics

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier that can be configured for use as an on-chip oscillator, as shown in Figure 11-1. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven, as shown in Figure 11-2. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clock- ing circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage high and low time specifications must be observed.

Figure 11-1. Oscillator Connections

Note: C1, C2 = 30 pF ± 10 pF for Crystals
      = 40 pF ± 10 pF for Ceramic Resonators
2. Idle Mode

In idle mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The content of the on-chip RAM and all the special function registers remain unchanged during this mode. The idle mode can be terminated by any enabled interrupt or by a hardware reset.

Note that when idle mode is terminated by a hardware reset, the device normally resumes program execution from where it left off, in 1 to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write to a port pin when idle mode is terminated by a reset, the instruction following the one that invokes idle mode should not write to a port pin or to external memory.

3. Power-down Mode

In the power-down mode, the oscillator is stopped, and the instruction that invokes power-down is the last instruction executed. The on-chip RAM and special function registers retain their values until the power-down mode is terminated. Exit from power-down mode can be initiated either by a hardware reset or by activation of an enabled external interrupt (INT0 or INT1). Reset redefines the SFRs but does not change the on-chip RAM. The reset should not be activated before $V_{CC}$ is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and stabilize.

<table>
<thead>
<tr>
<th>Table 13-1</th>
<th>Status of External Pins During Idle and Power-down Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Program Memory</td>
</tr>
<tr>
<td>Idle</td>
<td>Internal</td>
</tr>
<tr>
<td>Idle</td>
<td>External</td>
</tr>
<tr>
<td>Power-down</td>
<td>Internal</td>
</tr>
<tr>
<td>Power-down</td>
<td>External</td>
</tr>
</tbody>
</table>
4. Program Memory Lock Bits

The AT89S51 has three lock bits that can be left unprogrammed (U) or can be programmed (P) to obtain the additional features listed in Table 14-1.

<table>
<thead>
<tr>
<th>Program Lock Bits</th>
<th>LB1</th>
<th>LB2</th>
<th>LB3</th>
<th>Protection Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>No program lock features</td>
</tr>
<tr>
<td>2</td>
<td>P</td>
<td>U</td>
<td>U</td>
<td>MOVIC instructions executed from external program memory are disabled from locking code bytes from internal memory. EA is sampled and latched on reset, and further programming of the Flash memory is disabled.</td>
</tr>
<tr>
<td>3</td>
<td>P</td>
<td>P</td>
<td>U</td>
<td>Same as mode 2, but verify is also disabled.</td>
</tr>
<tr>
<td>4</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>Same as mode 3, but external execution is also disabled.</td>
</tr>
</tbody>
</table>

When lock bit 1 is programmed, the logic level at the EA pin is sampled and latched during reset. If the device is powered up without a reset, the latch initializes to a random value and holds that value until reset is activated. The latched value of EA must agree with the current logic level at that pin in order for the device to function properly.

5. Programming the Flash – Parallel Mode

The AT89S51 is shipped with the on-chip Flash memory array ready to be programmed. The programming interface needs a high-voltage (12-volt) program enable signal and is compatible with conventional third-party Flash or EPROM programmers.

The AT89S51 1004 memory array is programmed byte-by-byte.

Programming Algorithm: Before programming the AT89S51, the address, data, and control signals should be set up according to the Flash Programming Modes table (Table 17-1) and Figure 17-1 and Figure 17-2. To program the AT89S51, take the following steps:

1. Input the desired memory location on the address lines.
2. Input the appropriate data byte on the data lines.
3. Activate the correct combination of control signals.
4. Raise EA/Vpp to 12V.
5. Pulse ALE/PROG once to program a byte in the Flash array or the lock bits. The byte-write cycle is self-timed and typically takes no more than 30 µs. Repeat steps 1 through 5, changing the address and data for the entire array or until the end of the object file is reached.

Data Polling: The AT89S51 features Data Polling to indicate the end of a byte write cycle. During a write cycle, an attempted read of the last byte written will result in the complement of the written data on P0.7. Once the write cycle has been completed, true data is valid on all outputs, and the next cycle may begin. Data Polling may begin any time after a write cycle has been initiated.

Ready/Rdy: The progress of byte programming can also be monitored by the RDY/BSY output signal. P0.0 is pulled low after ALE goes high during programming to indicate BSY. P3.0 is pulled high again when programming is done to indicate READY.
Program Verify: If lock bits LB1 and LB2 have not been programmed, the programmed code data can be read back via the address and data lines for verification. The status of the individual lock bits can be verified directly by reading them back.

Reading the Signature Bytes: The signature bytes are read by the same procedure as a normal verification of locations 000H, 100H, and 200H, except that P3.6 and P3.7 must be pulled to a logic low. The values returned are as follows.

\begin{align*}
(000H) &= 1EH \\
(100H) &= 51H \\
(200H) &= 08H
\end{align*}

Chip Erase: In the parallel programming mode, a chip erase operation is initiated by using the proper combination of control signals and by pulsing ALE/PROG low for a duration of 200 ns - 500 ns.

In the serial programming mode, a chip erase operation is initiated by issuing the Chip Erase instruction. In this mode, chip erase is self-timed and takes about 500 ns.

During chip erase, a serial read from any address location will return 00H at the data output.

6. Programming the Flash – Serial Mode

The Code memory array can be programmed using the serial SPI interface while RST is pulled to Vcc. The serial interface consists of pins SCK, MOSI (input) and MISO (output). After RST is set high, the Programming Enable instruction needs to be executed first before other operations can be executed. Before a reprogramming sequence can occur, a Chip Erase operation is required.

The Chip Erase operation turns the content of every memory location in the Code array into FFH.

Either an external crystal clock can be supplied at pin XTAL1 or a crystal needs to be connected across pins XTAL1 and XTAL2. The maximum serial clock (SCK) frequency should be less than 1/16 of the crystal frequency. With a 33 MHz oscillator clock, the maximum SCK frequency is 2 MHz.

5.1 Serial Programming Algorithm

To program and verify the AT89S51 in the serial programming mode, the following sequence is recommended:

1. Power-up sequence:
   a. Apply power between VCC and GND pins.
   b. Set RST pin to "H".

2. If a crystal is not connected across pins XTAL1 and XTAL2, apply a 3 MHz to 33 MHz clock to XTAL1 pin and wait for at least 10 milliseconds.

3. Enable serial programming by sending the Programming Enable serial instruction to pin MOSIP1.5. The frequency of the shift clock switched at pin SCKP1.7 needs to be less than the CPU clock at XTAL1 divided by 16.

4. The Code array is programmed one byte at a time in either the Byte or Page mode. The write cycle is self-timed and typically takes less than 0.5 ms at 5V.
5. At the end of a programming session, RST can be set low to commence normal device operation.

Power-off sequence (if needed):
1. Set XTAL1 to "L" (if a crystal is not used).
2. Set RST to "L".
3. Turn VCC power off.

Data Polling: The Data Polling feature is also available in the serial mode. In this mode, during a write cycle an attempted read of the last byte written will result in the complement of the MSB of the serial output byte on MISO.

5.2 Serial Programming Instruction Set

The Instruction Set for Serial Programming follows a 4-byte protocol and is shown in the “Serial Programming Instruction Set” on page 20.

7. Programming Interface – Parallel Mode

Every code byte in the Flash array can be programmed by using the appropriate combination of control signals. The write operation cycle is set-timed and once initiated, will automatically time itself to completion.

Most major worldwide programming vendors offer worldwide support for the Atmel AT89 microcontroller series. Please contact your local programming vendor for the appropriate software revision.

<table>
<thead>
<tr>
<th>Mode</th>
<th>VCC</th>
<th>RST</th>
<th>PSEN</th>
<th>ALE/PROG</th>
<th>EA/VP</th>
<th>P2.6</th>
<th>P2.7</th>
<th>P1.3</th>
<th>P3.6</th>
<th>P3.7</th>
<th>P4.7-0</th>
<th>P5.3-0</th>
<th>P6.7-0</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write Code Data 9V</td>
<td>L</td>
<td>L</td>
<td></td>
<td></td>
<td>12V</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>D49</td>
<td>A11-0</td>
<td>A7-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read Code Data   9V</td>
<td>L</td>
<td>L</td>
<td></td>
<td></td>
<td>12V</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write Lock Bit 1 9V</td>
<td>L</td>
<td>L</td>
<td></td>
<td></td>
<td>12V</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write Lock Bit 2 9V</td>
<td>L</td>
<td>L</td>
<td></td>
<td></td>
<td>12V</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read Lock Bits 1, 2, 3 9V</td>
<td>L</td>
<td>L</td>
<td></td>
<td></td>
<td>12V</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chip Erase      9V</td>
<td>L</td>
<td>L</td>
<td></td>
<td></td>
<td>12V</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes:
1. Each PROG pulse is 200 ns - 500 ns for Chip Erase.
2. Each PROG pulse is 200 ns - 500 ns for Write Code Data.
3. Each PROG pulse is 200 ns - 500 ns for Write Lock Bits.
4. RDY/BUSY signal is output on P3.6 during programming.
5. X = don't care.

AT89S51
### 8. Flash Programming and Verification Characteristics (Parallel Mode)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter Description</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_{PP} )</td>
<td>Programming Supply Voltage</td>
<td>11.5</td>
<td>12.5</td>
<td>V</td>
</tr>
<tr>
<td>( I_{PP} )</td>
<td>Programming Supply Current</td>
<td>10</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>( I_{CC} )</td>
<td>( V_{CC} ) Supply Current</td>
<td>30</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>( f_{Osc} )</td>
<td>Oscillator Frequency</td>
<td>3</td>
<td>33</td>
<td>MHz</td>
</tr>
<tr>
<td>( t_{ASL} )</td>
<td>Address Setup to PROG Low</td>
<td>48 ( t_{CL,CL} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{ASH} )</td>
<td>Address Hold after PROG</td>
<td>48 ( t_{CL,CL} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{DSL} )</td>
<td>Data Setup to PROG Low</td>
<td>48 ( t_{CL,CL} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{DSH} )</td>
<td>Data Hold after PROG</td>
<td>48 ( t_{CL,CL} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{P2.7, (ENABLE), High, to, V_{PP} } )</td>
<td></td>
<td>48 ( t_{CL,CL} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{VPP, Setup, to, PROG, Low} )</td>
<td>10</td>
<td></td>
<td>( \mu\text{s} )</td>
<td></td>
</tr>
<tr>
<td>( t_{V_{PP}, Hold, after, PROG} )</td>
<td>10</td>
<td></td>
<td>( \mu\text{s} )</td>
<td></td>
</tr>
<tr>
<td>( t_{P}_{CHM} )</td>
<td>PROG Width</td>
<td>0.2</td>
<td>1</td>
<td>( \mu\text{s} )</td>
</tr>
<tr>
<td>( t_{AVC} )</td>
<td>Address to Data Valid</td>
<td>48 ( t_{CL,CL} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{EAV} )</td>
<td>ENABLE Low to Data Valid</td>
<td>48 ( t_{CL,CL} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{DF} )</td>
<td>Data Float After ENABLE</td>
<td>0</td>
<td>48 ( t_{CL,CL} )</td>
<td></td>
</tr>
<tr>
<td>( t_{P2.7, (ENABLE), PROG, High, to, BUSY, Low} )</td>
<td></td>
<td>1.0</td>
<td></td>
<td>( \mu\text{s} )</td>
</tr>
<tr>
<td>( t_{W} )</td>
<td>Strobe Write Cycle Time</td>
<td>50</td>
<td></td>
<td>( \mu\text{s} )</td>
</tr>
</tbody>
</table>

---

**Figure 18-1.** Flash Programming and Verification Waveforms – Parallel Mode

![Waveforms Diagram](image-url)
Figure 18-2. Flash Memory Serial Downloading

Figure 19-1. Serial Programming Waveforms

9. Flash Programming and Verification Waveforms - Serial Mode
0. Serial Programming Instruction Set

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Enable</td>
<td>1010</td>
<td>1100</td>
<td>0101 0011</td>
<td>xxxxxxx</td>
<td>Enable Serial Programming when RST is high</td>
</tr>
<tr>
<td>Chip Erase</td>
<td>1010</td>
<td>1100</td>
<td>1000 xxx</td>
<td>xxxxxxx</td>
<td>Chip Erase Flash memory array</td>
</tr>
<tr>
<td>Read Program Memory (Byte Mode)</td>
<td>0100</td>
<td>0000</td>
<td>xxxxxxx</td>
<td>xxxxxxx</td>
<td>Read data from Program memory in the byte mode</td>
</tr>
<tr>
<td>Write Program Memory (Byte Mode)</td>
<td>1010</td>
<td>1100</td>
<td>1110 0101</td>
<td>xxxxxxx</td>
<td>Write data to Program memory in the byte mode</td>
</tr>
<tr>
<td>Write Lock Bits</td>
<td>0100</td>
<td>0100</td>
<td>xxxxxxx</td>
<td>xxxxxxx</td>
<td>Write Lock bits. See Note (1).</td>
</tr>
<tr>
<td>Read Lock Bits</td>
<td>0010</td>
<td>1000</td>
<td>xxxxxxx</td>
<td>xxxxxxx</td>
<td>Read back current status of the lock bits (a programmed lock bit reads back as a &quot;1&quot;)</td>
</tr>
<tr>
<td>Read Signature Bytes</td>
<td>0010</td>
<td>1000</td>
<td>xxxxxxx</td>
<td>xxxxxxx</td>
<td>Read Signature Byte</td>
</tr>
<tr>
<td>Read Program Memory (Page Mode)</td>
<td>0101</td>
<td>0000</td>
<td>xxxxxxx</td>
<td>xxxxxxx</td>
<td>Read data from Program memory in the Page Mode (256 bytes)</td>
</tr>
<tr>
<td>Write Program Memory (Page Mode)</td>
<td>0101</td>
<td>0000</td>
<td>xxxxxxx</td>
<td>xxxxxxx</td>
<td>Write data to Program memory in the Page Mode (256 bytes)</td>
</tr>
</tbody>
</table>

Notes:
1. B1 = 0, B2 = 0 → Mode 1, no lock protection
2. B1 = 0, B2 = 1 → Mode 2, lock bit 1 activated
3. B1 = 1, B2 = 0 → Mode 3, lock bit 2 activated
4. B1 = 1, B2 = 1 → Mode 4, lock bit 3 activated

Each of the lock bit modes need to be activated sequentially before Mode 4 can be executed.

Ter Reset signal is high, SCK should be low for at least 64 system clocks before it goes high to clock in the enable data. No pulsing of Reset signal is necessary. SCK should be no faster than 1/16 of the system clock at XTAL.

For Page Read/Write, the data always starts from byte 0 to 255. After the command byte and upper address byte are latched, each byte thereafter is treated as data until at 256 bytes as shifted in/out. Then the next instruction will be ready to be decoded.

AT89S51
1. Serial Programming Characteristics

![Serial Programming Timing Diagram]

Table 21-1. Serial Programming Characteristics, $T_x = -40^\circ C$ to $85^\circ C$, $V_{DD} = 4.0$ - $5.5V$ (Unless Otherwise Noted)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{osc}$</td>
<td>Oscillator Frequency</td>
<td>8</td>
<td>MHz</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>$t_{CL}$</td>
<td>Oscillator Period</td>
<td>30</td>
<td>ns</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>$t_{HIGH}$</td>
<td>SCK Pulse Width High</td>
<td>$t_{CL}$</td>
<td>ns</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>$t_{LOW}$</td>
<td>SCK Pulse Width Low</td>
<td>$t_{CL}$</td>
<td>ns</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>$t_{HIGH}$</td>
<td>MOSI Setup to SCK High</td>
<td>$t_{CL}$</td>
<td>ns</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>$t_{HIGH}$</td>
<td>MOSI Hold after SCK High</td>
<td>$2t_{CL}$</td>
<td>ns</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>$t_{LOW}$</td>
<td>SCK Low to MISO Valid</td>
<td>10</td>
<td>16</td>
<td>32</td>
<td>ns</td>
</tr>
<tr>
<td>$t_{ERASE}$</td>
<td>Chip Erase Instruction Cycle Time</td>
<td>500</td>
<td>ms</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>$t_{WRITE}$</td>
<td>Serial Byte Write Cycle Time</td>
<td>$64t_{CL}$</td>
<td>400</td>
<td>$\mu$s</td>
<td></td>
</tr>
</tbody>
</table>

2. Absolute Maximum Ratings*

- Operating Temperature: $-65^\circ C$ to $+125^\circ C$
- Storage Temperature: $-65^\circ C$ to $+150^\circ C$
- Voltage on Any Pin with Respect to Ground: $1.0V$ to $7.0V$
- Maximum Operating Voltage: $6.6V$
- $I_C$ Output Current: $15.0 \text{ mA}$

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.
# 3. DC Characteristics

The values shown in this table are valid for $T_A = -40°C$ to $85°C$ and $V_{CC} = 4.0V$ to $5.5V$, unless otherwise noted.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Condition</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{ss}$</td>
<td>Input Low Voltage</td>
<td>(Except $EA$)</td>
<td>-0.5</td>
<td>0.2</td>
<td>$V_{CC} - 0.1$</td>
</tr>
<tr>
<td>$I_{SS}$</td>
<td>Input Low Voltage ($EA$)</td>
<td></td>
<td>-0.5</td>
<td>0.2</td>
<td>$V_{CC} - 0.3$</td>
</tr>
<tr>
<td>$I_H$</td>
<td>Input High Voltage</td>
<td>(Except $XTAL1, RST$)</td>
<td>0.2</td>
<td>$V_{CC} - 0.9$</td>
<td></td>
</tr>
<tr>
<td>$I_{IH}$</td>
<td>Input High Voltage</td>
<td>($XTAL1, RST$)</td>
<td>0.7</td>
<td>$V_{CC}$</td>
<td></td>
</tr>
<tr>
<td>$I_{OL}$</td>
<td>Output Low Voltage</td>
<td>(Ports 1,2,3)</td>
<td></td>
<td>$I_{OL} = 1.6 mA$</td>
<td></td>
</tr>
<tr>
<td>$I_{OL}$</td>
<td>Output Low Voltage</td>
<td></td>
<td></td>
<td>$I_{OL} = 3.2 mA$</td>
<td></td>
</tr>
<tr>
<td>$I_{OH}$</td>
<td>Output High Voltage</td>
<td>(Ports 1,2,3, ALE, PB2, PB3)</td>
<td></td>
<td>$I_{OH} = 40 \mu A, V_{CC} = 5V ± 10%$</td>
<td></td>
</tr>
<tr>
<td>$I_{OH}$</td>
<td>Output High Voltage</td>
<td>(Ports 0 in External Bus Mode)</td>
<td></td>
<td>$I_{OH} = 600 \mu A, V_{CC} = 5V ± 10%$</td>
<td></td>
</tr>
<tr>
<td>$I_{IL}$</td>
<td>Input Leakage Current</td>
<td>(Port 0, $EA$)</td>
<td></td>
<td>$I_{IL} = 60 \mu A$</td>
<td></td>
</tr>
<tr>
<td>$I_{io}$</td>
<td>Logical 0 Input Current</td>
<td>(Ports 1,2,3)</td>
<td></td>
<td>$I_{io} = 0.45V$</td>
<td></td>
</tr>
<tr>
<td>$I_{in}$</td>
<td>Logical 1 to 0 Transition Current</td>
<td>(Ports 1,2,3)</td>
<td></td>
<td>$V_{IN} = 2V, V_{CC} = 5V ± 10%$</td>
<td></td>
</tr>
<tr>
<td>$I_{IL}$</td>
<td>Input Leakage Current</td>
<td>(Port 0, $EA$)</td>
<td></td>
<td>$I_{IL} = 60 \mu A$</td>
<td></td>
</tr>
<tr>
<td>$R_{FIRST}$</td>
<td>Relay Pulldown Resistor</td>
<td></td>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>$C_D$</td>
<td>Pullup Capacitance</td>
<td></td>
<td></td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>$CC$</td>
<td>Power Supply Current</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$CC$</td>
<td>Active Mode, 12 MHz</td>
<td></td>
<td></td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>$CC$</td>
<td>Idle Mode, 12 MHz</td>
<td></td>
<td></td>
<td>6.5</td>
<td></td>
</tr>
<tr>
<td>$CC$</td>
<td>Power-down Mode$^{(2)}$</td>
<td></td>
<td></td>
<td>$V_{CC} = 5.5V$</td>
<td></td>
</tr>
</tbody>
</table>

Notes:

1. Under steady state (non-transient) conditions, $I_{oH}$ must be externally limited as follows:
   - Maximum $I_{oH}$ per port is 10 mA
   - Maximum $I_{oH}$ per port per pin is 8-bit ports 10 mA
   - Maximum $I_{oH}$ for all output pins 21 mA
   - If $I_{oH}$ exceeds the test condition, $V_{DC}$ may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test conditions.

2. Minimum $V_{CC}$ for Power-down is 2V.
## 4. AC Characteristics

Under operating conditions, load capacitance for Port 0, ALE/PROG, and PSENI = 100 pF; load capacitance for all other Pins = 80 pF.

### 1. External Program and Data Memory Characteristics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>12 MHz Oscillator</th>
<th>Variable Oscillator</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCLR</td>
<td>Oscillator Frequency</td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
</tr>
<tr>
<td>LPL</td>
<td>ALE Pulse Width</td>
<td>127</td>
<td>2 LPL. 40</td>
<td>ns</td>
</tr>
<tr>
<td>AVL</td>
<td>Address Valid to ALE Low</td>
<td>48</td>
<td>1 AVL 25</td>
<td>ns</td>
</tr>
<tr>
<td>LAX</td>
<td>Address Hold After ALE Low</td>
<td>48</td>
<td>1 LAX 25</td>
<td>ns</td>
</tr>
<tr>
<td>LVE</td>
<td>ALE Low to Valid Instruction In</td>
<td>233</td>
<td>4 LVE. 65</td>
<td>ns</td>
</tr>
<tr>
<td>LPLI</td>
<td>ALE Low to PSN Low</td>
<td>43</td>
<td>1 LPL 25</td>
<td>ns</td>
</tr>
<tr>
<td>PLH</td>
<td>PSN Pulse Width</td>
<td>205</td>
<td>3 PLH. 45</td>
<td>ns</td>
</tr>
<tr>
<td>PLM</td>
<td>PSN Low to Valid Instruction In</td>
<td>141</td>
<td>3 PLM. 90</td>
<td>ns</td>
</tr>
<tr>
<td>PTH</td>
<td>Input Instruction Hold After PSN</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>PIH</td>
<td>Input Instruction Float After PSN</td>
<td>59</td>
<td>1 PIH 25</td>
<td>ns</td>
</tr>
<tr>
<td>PHA</td>
<td>PSN to Address Valid</td>
<td>75</td>
<td>1 PHA 4</td>
<td>ns</td>
</tr>
<tr>
<td>PVH</td>
<td>Address to Valid Instruction In</td>
<td>312</td>
<td>5 PVH. 90</td>
<td>ns</td>
</tr>
<tr>
<td>PLZ</td>
<td>PSN Low to Address Float</td>
<td>10</td>
<td>10</td>
<td>ns</td>
</tr>
<tr>
<td>RH</td>
<td>PSR Pulse Width</td>
<td>406</td>
<td>6 RH. 200</td>
<td>ns</td>
</tr>
<tr>
<td>WPH</td>
<td>WR Pulse Width</td>
<td>900</td>
<td>6 WPH. 200</td>
<td>ns</td>
</tr>
<tr>
<td>RLV</td>
<td>RD Low to Valid Data In</td>
<td>252</td>
<td>5 RLV. 90</td>
<td>ns</td>
</tr>
<tr>
<td>RNH</td>
<td>Data Hold After RD</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>RSD</td>
<td>Data Select After RD</td>
<td>97</td>
<td>2 RSD. 90</td>
<td>ns</td>
</tr>
<tr>
<td>RLW</td>
<td>ALE Low to Valid Data In</td>
<td>517</td>
<td>8 RLW. 190</td>
<td>ns</td>
</tr>
<tr>
<td>AOV</td>
<td>Address to Valid Data In</td>
<td>586</td>
<td>9 AOV 165</td>
<td>ns</td>
</tr>
<tr>
<td>LIM</td>
<td>ALE Low to RD or WR Low</td>
<td>200</td>
<td>3 LIM. 50</td>
<td>3 LIM. 50</td>
</tr>
<tr>
<td>CRM</td>
<td>Address to RD or WR Low</td>
<td>203</td>
<td>4 CRM. 75</td>
<td>ns</td>
</tr>
<tr>
<td>SW</td>
<td>Data Valid to WR Transition</td>
<td>23</td>
<td>1 SW 25</td>
<td>ns</td>
</tr>
<tr>
<td>SWH</td>
<td>Data Valid to WR High</td>
<td>433</td>
<td>7 SWH. 130</td>
<td>ns</td>
</tr>
<tr>
<td>SMD</td>
<td>Data Hold After WR</td>
<td>33</td>
<td>1 SMD 25</td>
<td>ns</td>
</tr>
<tr>
<td>PLZL</td>
<td>RD Low to Address Float</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>WDI</td>
<td>RD or WR High to ALE High</td>
<td>43</td>
<td>1 WDI 25</td>
<td>1 WDI 25</td>
</tr>
</tbody>
</table>
5. External Program Memory Read Cycle

6. External Data Memory Read Cycle

AT89S51
7. External Data Memory Write Cycle

![Diagram of External Data Memory Write Cycle]

8. External Clock Drive Waveforms

![Diagram of External Clock Drive Waveforms]

9. External Clock Drive

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOSC</td>
<td>Oscillator Frequency</td>
<td>0</td>
<td>33</td>
<td>kHz</td>
</tr>
<tr>
<td>TCKL</td>
<td>Clock Period</td>
<td>30</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TCKH</td>
<td>High Time</td>
<td>12</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TCKL</td>
<td>Low Time</td>
<td>12</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TCRL</td>
<td>Rise Time</td>
<td>5</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TCCL</td>
<td>Fall Time</td>
<td>5</td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>
0. Serial Port Timing: Shift Register Mode Test Conditions

All values in this table are valid for \( V_{CC} = 4.0 \text{V} \) to 5.6\text{V} and Load Capacitance = 80\text{pF}.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>12 MHz Osc</th>
<th>Variable Oscillator</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_{XZ} )</td>
<td>Serial Port Clock Cycle Time</td>
<td>Min 1.0</td>
<td>Max ( \tau_{XZ} )</td>
</tr>
<tr>
<td>( \tau_{XH} )</td>
<td>Output Data Setup to Clock Rising Edge</td>
<td>Min 700</td>
<td>Max ( \tau_{XH} )</td>
</tr>
<tr>
<td>( \tau_{XQ} )</td>
<td>Output Data Hold After Clock Rising Edge</td>
<td>Min 50</td>
<td>Max ( \tau_{XQ} )</td>
</tr>
<tr>
<td>( \tau_{XH} )</td>
<td>Input Data Hold After Clock Rising Edge</td>
<td>Min 0</td>
<td>Max ( \tau_{XH} )</td>
</tr>
<tr>
<td>( \tau_{XH} )</td>
<td>Clock Rising Edge to Input Data Valid</td>
<td>Min 700</td>
<td>Max ( \tau_{XH} )</td>
</tr>
</tbody>
</table>

1. Shift Register Mode Timing Waveforms

2. AC Testing Input/Output Waveforms\(^{(1)}\)

\( \text{At:} \) 1. AC inputs during testing are driven at \( V_{CC} = 0.5 \text{V} \) for a logic 1 and 0.45\text{V} for a logic 0. Timing measurements are made at \( V_{CC} \) min for a logic 1 and \( V_{CC} \) max for a logic 0.

3. Float Waveforms\(^{(1)}\)

\( \text{At:} \) 1. For timing purposes, a port pin is no longer floating when a 100 m\text{V} change from load voltage occurs. A port pin begins to float when a 100 m\text{V} change of the loaded \( V_{DD} / V_{OL} \) level occurs.

---

AT89S51

04/3C-Micro-0305
4. Ordering Information

## 4.1 Standard Package

<table>
<thead>
<tr>
<th>Speed (MHz)</th>
<th>Power Supply</th>
<th>Ordering Code</th>
<th>Package</th>
<th>Operation Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>4.0V to 5.5V</td>
<td>AT89S51-24AC</td>
<td>44A</td>
<td>Commercial (0°C to 70°C)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89S51-24JC</td>
<td>44J</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89S51-24PC</td>
<td>40PS6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89S51-24SC</td>
<td>42PS6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89S51-24AI</td>
<td>44A</td>
<td>Industrial (-40°C to 85°C)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89S51-24JI</td>
<td>44J</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89S51-24PI</td>
<td>40PS6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89S51-24SI</td>
<td>42PS6</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>4.5V to 5.5V</td>
<td>AT89S51-33AC</td>
<td>44A</td>
<td>Commercial (0°C to 70°C)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89S51-33JC</td>
<td>44J</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89S51-33PC</td>
<td>40PS6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89S51-33SC</td>
<td>42PS6</td>
<td></td>
</tr>
</tbody>
</table>

## 4.2 Green Package Option (Pb/Halide-free)

<table>
<thead>
<tr>
<th>Speed (MHz)</th>
<th>Power Supply</th>
<th>Ordering Code</th>
<th>Package</th>
<th>Operation Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>4.0V to 5.5V</td>
<td>AT89S51-24AU</td>
<td>44A</td>
<td>Industrial (-40°C to 85°C)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89S51-24JU</td>
<td>44J</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89S51-24PU</td>
<td>40PS6</td>
<td></td>
</tr>
</tbody>
</table>

### Package Type

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>44A</td>
<td>44-lead, Thin Plastic Quad Flatpack (TOFP)</td>
</tr>
<tr>
<td>44J</td>
<td>44-lead, Plastic J-Leaded Chip Carrier (PLCC)</td>
</tr>
<tr>
<td>40P6</td>
<td>40-pin, 0.600&quot; Wide, Plastic Dual In-line Package (PDP)</td>
</tr>
<tr>
<td>42PS6</td>
<td>42-pin, 0.600&quot; Wide, Plastic Dual In-line Package (PDP)</td>
</tr>
</tbody>
</table>
5. Packaging Information

5.1 44A - TQFP

![Diagram of 44A - TQFP package]

**Notes:**
1. This package conforms to JEDEC reference MS-026B, Variation AC8.
2. Dimensions D1 and E1 do not include mold projection. The maximum mold projection is 0.25 mm per side. Dimensions D1 and E1 are maximum plastic body dimensions including mold mismatch.
3. Lead coplanarity is 0.10 mm maximum.

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>MIN</th>
<th>NOM</th>
<th>MAX</th>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-</td>
<td>-</td>
<td>1.20</td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>0.05</td>
<td>-</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>0.95</td>
<td>1.00</td>
<td>3.05</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>11.75</td>
<td>12.00</td>
<td>12.25</td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td>9.90</td>
<td>10.00</td>
<td>10.10</td>
<td>Note 2</td>
</tr>
<tr>
<td>E</td>
<td>11.75</td>
<td>12.00</td>
<td>12.25</td>
<td></td>
</tr>
<tr>
<td>E1</td>
<td>9.90</td>
<td>10.00</td>
<td>10.10</td>
<td>Note 2</td>
</tr>
<tr>
<td>B</td>
<td>0.38</td>
<td>-</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0.09</td>
<td>-</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>0.45</td>
<td>-</td>
<td>0.75</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>0.80</td>
<td>TYP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2325 Orchard Parkway
San Jose, CA 95131

**TITLE:** 44A, 44-lead, 10 x 10 mm Body Size, 1.0 mm Body Thickness, 0.8 mm Lead Pitch, Thin Profile Plastic Quad Flat Pack (TQFP)

**DRAWING NO.:** 44A

**REV.:** B

12/5/2001

AT89S51
Lampiran 2:

Data Sheet LCD
Unit: mm/ inch
General tolerance: ±0.5 mm

<table>
<thead>
<tr>
<th>No.</th>
<th>Symbol</th>
<th>Level</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Vcc</td>
<td></td>
<td>Power Supply</td>
</tr>
<tr>
<td>2</td>
<td>Vss</td>
<td></td>
<td>Power GND</td>
</tr>
<tr>
<td>3</td>
<td>Vcc</td>
<td></td>
<td>for LED Drive</td>
</tr>
<tr>
<td>4</td>
<td>RS</td>
<td>H/L</td>
<td>H: Data Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L: Instruction Input</td>
</tr>
<tr>
<td>5</td>
<td>R/W</td>
<td>H/L</td>
<td>H: READ L/WRITE</td>
</tr>
<tr>
<td>6</td>
<td>E</td>
<td>H/L</td>
<td>Enable Signal</td>
</tr>
<tr>
<td>7</td>
<td>DD0</td>
<td>H/L</td>
<td>Data Bus</td>
</tr>
<tr>
<td>8</td>
<td>DD1</td>
<td>H/L</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>DD2</td>
<td>H/L</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>DD3</td>
<td>H/L</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>DD4</td>
<td>H/L</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>DD5</td>
<td>H/L</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>DD6</td>
<td>H/L</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>DD7</td>
<td>H/L</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>V+BL</td>
<td></td>
<td>Back Light 50-200mA</td>
</tr>
<tr>
<td>16</td>
<td>V-GL</td>
<td></td>
<td>Supply 4.2V GND</td>
</tr>
</tbody>
</table>

Figure 1 Dimensions diagram
OPERATING INSTRUCTIONS

INTRODUCTION

Seiko Instruments intelligent dot matrix liquid crystal display modules have on-board controller and LSI drivers, which display alpha numerics, Japanese KATA KANA characters and a wide variety of other symbols in either 5 x 7 dot matrix.

The internal operation in the KS0056 controller chip is determined by signals sent from the MPU. The signals include: 1) Register select RS input consisting of instruction register (IR) when RS = 0 and data register (DR) when RS = 1; 2) Read/write (RW); 3) Data bus (DB7-DB0); and 4) Enable strobe (E) depending on the MPU or through an external parallel I/O port. Details on instructions data entry, execution times, etc. are explained in the following sections.

READ AND WRITE TIMING DIAGRAMS AND TABLES

The following timing characteristics are applicable for all of Seiko's LCD dot matrix character modules.

### READ TIMING CHARACTERISTICS

<table>
<thead>
<tr>
<th>Item</th>
<th>Symbol</th>
<th>Standard</th>
<th>Unit</th>
<th>Min.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable cycle time</td>
<td>t_{EN}</td>
<td>500</td>
<td>ns</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Enable pulse High Level</td>
<td>PW_{EN}</td>
<td>300</td>
<td>ns</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Enable rise and fall time</td>
<td>t_{rL}</td>
<td>20</td>
<td>ns</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Address setup time</td>
<td>t_{AD}</td>
<td>40</td>
<td>ns</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Address hold time</td>
<td>t_{ADH}</td>
<td>10</td>
<td>ns</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Data delay time</td>
<td>t_{DD}</td>
<td>150</td>
<td>ns</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Data hold time</td>
<td>t_{DH}</td>
<td>5</td>
<td>ns</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### WRITE TIMING CHARACTERISTICS

<table>
<thead>
<tr>
<th>Item</th>
<th>Symbol</th>
<th>Standard</th>
<th>Unit</th>
<th>Min.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable cycle time</td>
<td>t_{EN}</td>
<td>500</td>
<td>ns</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Enable pulse High Level</td>
<td>PW_{EN}</td>
<td>200</td>
<td>ns</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Enable rise and fall time</td>
<td>t_{rL}</td>
<td>70</td>
<td>ns</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Address setup time</td>
<td>t_{AD}</td>
<td>40</td>
<td>ns</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Address hold time</td>
<td>t_{ADH}</td>
<td>10</td>
<td>ns</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Data setup time</td>
<td>t_{DS}</td>
<td>80</td>
<td>ns</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Data hold time</td>
<td>t_{DH}</td>
<td>10</td>
<td>ns</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
# Operating Instructions

## Introduction Codes

| Function | Code | Description | Execution Time
|----------|------|-------------|----------------|
| Display On/Off | 0 | Turn display on/off | 0.5 µs
| Character Generator | 1 | Display characters | 1 µs
| Status Register | 2 | Display status | 2 µs
| Data Input | 3 | Input data | 3 µs
| Data Output | 4 | Output data | 4 µs
| Data Input & Output | 5 | Input and output | 5 µs
| Command | 6 | Execute command | 6 µs
| Data Input & Command | 7 | Input and command | 7 µs

**Execution Times**
- 0.5 µs
- 1 µs
- 2 µs
- 3 µs
- 4 µs
- 5 µs
- 6 µs
- 7 µs

**Execution Time Table**

| Function | Code | Description | Execution Time
|----------|------|-------------|----------------|
| Clear Display | 0 | Clear display memory and return to home position | 82.5 µs
| Return Home | 0 | Return to home position | 40.4 µs
| Key Input Set | 0 | Set key input | 60.6 µs
| Display ON/OFF Control | 0 | Turn display on/off | 40.4 µs
| Current Display Shift | 0 | Display current shift | 40.4 µs
| Function Set | 0 | Set function | 40.4 µs
| Set E0 RAM Address | 0 | Set E0 RAM address | 40.4 µs
| Read E0 RAM & Address | 0 | Read E0 RAM & address | 40.4 µs
| Write Data to E0 RAM | 1 | Write data to E0 RAM | 49.5 µs
| Read Data from E0 RAM | 1 | Read data from E0 RAM | 49.5 µs

**Execution Times**
- 82.5 µs
- 40.4 µs
- 60.6 µs
- 40.4 µs
- 40.4 µs
- 40.4 µs
- 40.4 µs
- 40.4 µs
- 49.5 µs
- 49.5 µs

**Notes**
- Display data field
- CG RAM data
- Status register
- Command 1
- Display memory
- Status register
- Data input and output
- Command 2
- Input and output
- Command 3
- Data input and command
- Command 4
- Input and command
- Command 5
- Data input and output

**Additional Information**
- Execution times in the above table indicate the minimum values when operating frequency is 250 kHz.
- When $f_0$ is 270 kHz, $t_{DAC} = 2f_0 + 270$ µs
INTRODUCTION CODE EXPLANATIONS

The two registers 1) Instruction Register (IR) and the Data Register (DR) in the K5006 controller chip are directly operated by the MPU. Control information is temporarily stored in these registers prior to internal operation start. This allows interface to various types of MPUs which operate at different speeds from that of the K5006, and allows interface from peripheral control ICs. Internal operations of the K5006 are determined from the signals sent from the MPU. These signals, including register selection signals (RS), Read/Write (RW) and Status bus signals (DB5-DB0) are polled sequentially.

ADDRESS Counter (AC)
The counter specifies an address when data is written to DD RAM or CG RAM and the data stored in DD RAM or G RAM is read out. If an Address Set instruction (for DD RAM or CG RAM) is written in the IR, the address information transferred from the IR to the AC. When display data is written into or read from DD RAM or CG RAM, the AC is automatically incremented or decremented by one according to the Entry Mode Set. The contents of the AC are output to DSPP to DSPP+9 referring to the "Register Selection Table" when RS = 0 and RW = 1.

CLEAR DISPLAY

Code: 00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
OPERATING INSTRUCTIONS

ENTRY MODE $S_1$

<table>
<thead>
<tr>
<th>RS</th>
<th>KY</th>
<th>DB7</th>
<th>DB6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$S$: Increments ($S = 1$) or decrements ($S = 0$) the DD RAM address by one block when writing or reading a character code from DC RAM or CG RAM. The cursor automatically moves to the right when incremented by one or to the left if decremented by one.

DISPLAY AND CURSOR ON/OFF CONTROL

<table>
<thead>
<tr>
<th>RS</th>
<th>KY</th>
<th>DB7</th>
<th>DB6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$D$: Display is turned ON when $D = 1$ and OFF when $D = 0$. When display is OFF, display data in DD RAM remains unchanged. Information comes back immediately when $D = 1$ is inserted.

$C$: Cursor is displayed when $C = 1$ and not displayed when $C = 0$. If the cursor disappears, function of $S$ will not change during display data write in a 5 x 7 dot matrix there is an eighth line which functions as the cursor.

CURSOR OR DISPLAY SHIFT

<table>
<thead>
<tr>
<th>RS</th>
<th>KY</th>
<th>DB7</th>
<th>DB6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$S$: Shifts the entire display to either the right or left when $S = 1$ (right). When $S = 1$ and $D = 1$ the display shifts one position to the left. When $S = -1$ and $D = 0$ the display shifts one position to the right. This right or left shift occurs after each display write to DD RAM. Display is not shifted when result from DD RAM Display is not shifted when $S = 0$.

$C$: Does not change during display data write in a 5 x 7 dot matrix.

B: When $B = 1$, the character at the cursor position starts blinking. When $B = 0$ the cursor does not blink. The blink is done by blinking between the all black dot matrix and displayed character at 0.4 seconds intervals. The cursor and the blink can be set at the same time (basic = 50 Hz).

S X 7 DOT MATRIX

$C = 1$ (cursor display)

$B = 1$ (blinking)

CURSOR OR DISPLAY SHIFT

<table>
<thead>
<tr>
<th>RS</th>
<th>KY</th>
<th>DB7</th>
<th>DB6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Cursor/Display Shift moves the cursor or shifts the display without changing the DD RAM contents.

The cursor position and the AC contents match. This instruction is available for display correction and retrieval because the cursor position or display can be shifted without writing or reading display data. In case of a 2-line display, the cursor is shifted from character block 40 of line 1 to character block 1 of line 2. Displays of lines 1 and 2 are shifted at the same time. In case of a 4-line display, the cursor does not move continuously from line 2 to line 3. The cursor is shifted from character block 40 of line 3 to character block 1 of line 4. Displays of lines 3 and 4 are shifted at the same time. The display pattern of line 2 or 4 is not shifted to line 1 or 3.

<table>
<thead>
<tr>
<th>BIC</th>
<th>REL</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>The cursor position is shifted to the left (AC decrements one)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>The cursor position is shifted to the right (AC increments one)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>The entire display is shifted to the left with the cursor</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>The entire display is shifted to the right with the cursor</td>
</tr>
</tbody>
</table>
OPERATING INSTRUCTIONS

Function Set

<table>
<thead>
<tr>
<th>Code</th>
<th>Function Set</th>
<th><em>Don't Matter</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0010110</td>
<td>Don't Matter</td>
</tr>
</tbody>
</table>

DL: Interface data length
When DL = 1, the data length is set at 8 bits (DB7 to DB0).
When DL = 0, the data length is set at 4 bits (DB7 to DB4).
The upper 4 bits are transferred first, then the lower 4 bits follow.
N: Number of display lines
F: Select character font

<table>
<thead>
<tr>
<th>Number of display lines</th>
<th>Character font</th>
<th>Density</th>
<th>LCD Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5 x 7 dot matrix</td>
<td>196</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>All character LCD module</td>
<td></td>
</tr>
</tbody>
</table>

The function set instruction must be executed prior to all other instructions except Busy Flag Read. If another instruction is executed first, no function instruction except changing the interface data length can be executed.

CRAM ADDRESS SET

<table>
<thead>
<tr>
<th>Code</th>
<th>CRAM ADDRESS SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>00000000</td>
</tr>
</tbody>
</table>

CG RAM addresses, expressed as binary AAAAAA, are 0 to 15.

DD RAM ADDRESS SET

<table>
<thead>
<tr>
<th>Code</th>
<th>DD RAM ADDRESS SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>00000000</td>
</tr>
</tbody>
</table>

DD RAM addresses, expressed as binary AAAAAA, are 16 to 31.

BF/Flag/Address Read

<table>
<thead>
<tr>
<th>Code</th>
<th>BF/Flag/Address Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>00000000</td>
</tr>
</tbody>
</table>

BF signal is ready to verify if the controller is indicating that the module is working on a current instruction.
When BF = 1, the module is working internally and the next Instruction cannot be accepted until the BF value becomes 0.
When BF = 0, the next instruction can be accepted.

Therefore, make sure that BF = 0 before writing the next instruction. The AC values of binary AAAAAA are read out at the same time as reading the busy flag. The AC address is used for both CG RAM and DD RAM but the address set to an execution of the instruction determines which address to be used.

Data Write to CG RAM or DD RAM

<table>
<thead>
<tr>
<th>Code</th>
<th>Data Write to CG RAM or DD RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>00000000</td>
</tr>
</tbody>
</table>

Binary eight-bit data DDDD0000 is written into CG RAM or DD RAM. The CG RAM Address Set instruction or the DD RAM Address Set instruction before this instruction selects either FUM or FM. After that, the operation, the address and display shift are determined by the entry mode setting.

Data Read to CG RAM or DD RAM

<table>
<thead>
<tr>
<th>Code</th>
<th>Data Read to CG RAM or DD RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>00000000</td>
</tr>
</tbody>
</table>

Binary eight-bit data DDDD0000 is read from CG RAM or DD RAM. The CG RAM Address Set instruction or the DD RAM Address Set instruction before this instruction selects either RAM. In addition, either instruction is executed immediately before this instruction. If the Address Set instruction is executed before a read instruction, the last data read becomes invalid. If read instructions are executed consecutively, data is normally read from the second time. However, if the cursor is shifted by the Cursor Shift instruction when reading DD RAM, there is no need to write the address set instruction because the Cursor Shift instruction does this.

After the read operation, the address is automatically incremented or decremented by one according to the entry mode, but the display is not shifted.

Note: The AC is automatically incremented or decremented by one according to the entry mode after a write instruction is executed to write data to CG RAM or DD RAM. However, the data of the RAM selected by the AC are not read out even if a read instruction is executed immediately afterwards.
Operating Instructions

5 x 7 + Cursor

Relationships between CG RAM addresses and character codes (DD RAM) and character patterns (CG RAM data).

<table>
<thead>
<tr>
<th>Character Code (DD RAM data)</th>
<th>CG RAM address</th>
<th>Character pattern (CG RAM data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8BF</td>
<td>000 0</td>
<td>61</td>
</tr>
<tr>
<td>8BE</td>
<td>000 0</td>
<td>62</td>
</tr>
<tr>
<td>8BF</td>
<td>000 1</td>
<td>63</td>
</tr>
<tr>
<td>8BE</td>
<td>000 1</td>
<td>64</td>
</tr>
<tr>
<td>8BF</td>
<td>001 0</td>
<td>65</td>
</tr>
<tr>
<td>8BE</td>
<td>001 0</td>
<td>66</td>
</tr>
<tr>
<td>8BF</td>
<td>001 1</td>
<td>67</td>
</tr>
<tr>
<td>8BE</td>
<td>001 1</td>
<td>68</td>
</tr>
<tr>
<td>8BF</td>
<td>010 0</td>
<td>69</td>
</tr>
<tr>
<td>8BE</td>
<td>010 0</td>
<td>6A</td>
</tr>
<tr>
<td>8BF</td>
<td>010 1</td>
<td>6B</td>
</tr>
<tr>
<td>8BE</td>
<td>010 1</td>
<td>6C</td>
</tr>
<tr>
<td>8BF</td>
<td>011 0</td>
<td>6D</td>
</tr>
<tr>
<td>8BE</td>
<td>011 0</td>
<td>6E</td>
</tr>
<tr>
<td>8BF</td>
<td>011 1</td>
<td>6F</td>
</tr>
<tr>
<td>8BE</td>
<td>011 1</td>
<td>70</td>
</tr>
</tbody>
</table>

Example of character pattern (R)

Cursor position

Example of character pattern (Y)

Notes:
- In CG RAM data, 1 corresponds to Selection and 0 to Non-selection on the display.
- Character code bits 1 to 2 and CG RAM address bits 3 to 5 correspond with each other (three bits, eight types).
- CG RAM address bits 0 to 2 specify a line position for a character pattern. Line 8 of a character pattern is the cursor position where the logical sum of the cursor and CG RAM data is displayed. Set the data of line 8 to 0 to display the cursor. If the data is charged to 1, one bit light, regardless of the cursor.
- The same character code column position corresponds to CG RAM data bits 0 to 4 and to 4 comes to the left end. CG RAM data bits 5 to 7 are not displayed but can be used as general data RAM.
- When reading a character pattern from CG RAM, set to 0 all of character code bits 4 to 7. Bits 0 to 2 determine which pattern will be read out. Since bit 3 is not valid, 000 and 001 select the same character.
## Programing the Character Generator RAM (CG RAM)

The character generator RAM (CG RAM) allows the use of up to eight custom 5 x 7 characters or custom 8 x 8 characters. When programmed, the custom characters or symbols are stored exactly as if they were in ROM. However, since the CG RAM is a volatile memory, power must be continually supplied to it. Otherwise, the custom character symbols must be reprogrammed into non-volatile external ROM and sent to a display after each display initialization. All data in the 5 x 8 or 8 x 8 matrix can be programmed, which includes the cursor position.

The module RAM is divided into two parts: data store RAM (DDRAM) and custom character generator RAM (CG RAM). This is not to be confused with programming the custom character generator RAM with the 192 character locations, the first character of each row is MR 129. The CG RAM is located between MR 10 and MR 129 contiguous. Locations 40 thru 47 hold the first character, 48 thru 55 hold the second character, 56 thru 63 hold the third character, and so forth to 7F for the eighth. CG character symbols.

### Table: CG RAM Addresses

<table>
<thead>
<tr>
<th>RG</th>
<th>RW</th>
<th>Data</th>
<th>Display</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>40</td>
<td></td>
<td>addresses 1st row, 1st CG character</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>11</td>
<td></td>
<td>result of 0A, 2nd row</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0A</td>
<td></td>
<td>result of 0F, 3rd row</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>01</td>
<td>**</td>
<td>result of 04, 4th row</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>04</td>
<td>**</td>
<td>result of 0F, 5th row</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>04</td>
<td></td>
<td>result of 04, 6th row</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>64</td>
<td>**</td>
<td>result of 04, 7th row</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>00</td>
<td></td>
<td>result of 00, 8th row (cursor position)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>15</td>
<td>**</td>
<td>1st row, 2nd CG character, (Note: Addressing not now required; hex 44 is next in the sequence)</td>
</tr>
</tbody>
</table>

If during initialization the display with programmed characters is not initialized to a blank screen, then only the first initial address, 40, need be seen. Consecutive row data will automatically appear at 41, 42, etc. until the complete character is formed. All eight custom CG characters can be programmed in 64 consecutive write shifts sending the single initial 40 address.

The CG RAM is 8 bits wide, although only the right-most 5 bits are used for a custom CG character row. The left-most bit of programming the CG RAM character corresponds to the most significant nibble (000000) of the data bus code, with the remaining 3 bits in the row corresponding to the least significant nibble (03 thru 07), all being the right-most. Thus, hex FF equals all dots on and hex 00 equals all dots off. Examples include hex 15 (100101) equals 3 dots on the 50 CA (010101) equals 2 dots on. In each case the key 5 bits of the 8-bit code program one row of a custom CG character. When all 7 x 8 rows are programmed, the character is complete. A graphic example is shown below.
### Operating Instructions

#### Character Font Codes (5 x 7 dot matrix)

#### Upper 4 Bit Hexadecimal

<table>
<thead>
<tr>
<th>Upper 4 Bit Hexadecimal</th>
<th>0000 (0)</th>
<th>0010 (1)</th>
<th>0011 (2)</th>
<th>0100 (3)</th>
<th>0101 (4)</th>
<th>0110 (5)</th>
<th>0111 (6)</th>
<th>1010 (7)</th>
<th>1011 (8)</th>
<th>1100 (9)</th>
<th>1101 (10)</th>
<th>1110 (11)</th>
<th>1111 (12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>🅰️</td>
<td>🅱️</td>
<td>🅲️</td>
<td>🅳️</td>
<td>🅴️</td>
<td>🅵️</td>
<td>🅶️</td>
<td>🅷️</td>
<td>🅸️</td>
<td>🅹️</td>
<td>🅺️</td>
<td>🅻️</td>
<td>🅼️</td>
</tr>
<tr>
<td>0001</td>
<td>🅴️</td>
<td>🅵️</td>
<td>🅶️</td>
<td>🅷️</td>
<td>🅸️</td>
<td>🅹️</td>
<td>🅺️</td>
<td>🅻️</td>
<td>🅼️</td>
<td>🅽️</td>
<td>🅾️</td>
<td>🅿️</td>
<td>🅶️</td>
</tr>
<tr>
<td>0010</td>
<td>🅳️</td>
<td>🅴️</td>
<td>🅵️</td>
<td>🅶️</td>
<td>🅷️</td>
<td>🅸️</td>
<td>🅹️</td>
<td>🅺️</td>
<td>🅻️</td>
<td>🅼️</td>
<td>🅽️</td>
<td>🅾️</td>
<td>🅶️</td>
</tr>
<tr>
<td>0011</td>
<td>🅲️</td>
<td>🅳️</td>
<td>🅴️</td>
<td>🅵️</td>
<td>🅶️</td>
<td>🅷️</td>
<td>🅸️</td>
<td>🅹️</td>
<td>🅺️</td>
<td>🅻️</td>
<td>🅼️</td>
<td>🅽️</td>
<td>🅾️</td>
</tr>
<tr>
<td>0100</td>
<td>🅱️</td>
<td>🅲️</td>
<td>🅳️</td>
<td>🅴️</td>
<td>🅵️</td>
<td>🅶️</td>
<td>🅷️</td>
<td>🅸️</td>
<td>🅹️</td>
<td>🅺️</td>
<td>🅻️</td>
<td>🅼️</td>
<td>🅽️</td>
</tr>
<tr>
<td>0101</td>
<td>🅰️</td>
<td>🅱️</td>
<td>🅲️</td>
<td>🅳️</td>
<td>🅴️</td>
<td>🅵️</td>
<td>🅶️</td>
<td>🅷️</td>
<td>🅸️</td>
<td>🅹️</td>
<td>🅺️</td>
<td>🅻️</td>
<td>🅼️</td>
</tr>
<tr>
<td>0110</td>
<td>🅱️</td>
<td>🅲️</td>
<td>🅳️</td>
<td>🅴️</td>
<td>🅵️</td>
<td>🅶️</td>
<td>🅷️</td>
<td>🅸️</td>
<td>🅹️</td>
<td>🅺️</td>
<td>🅻️</td>
<td>🅼️</td>
<td>🅽️</td>
</tr>
<tr>
<td>0111</td>
<td>🅱️</td>
<td>🅲️</td>
<td>🅳️</td>
<td>🅴️</td>
<td>🅵️</td>
<td>🅶️</td>
<td>🅷️</td>
<td>🅸️</td>
<td>🅹️</td>
<td>🅺️</td>
<td>🅻️</td>
<td>🅼️</td>
<td>🅽️</td>
</tr>
<tr>
<td>1010</td>
<td>🅱️</td>
<td>🅲️</td>
<td>🅳️</td>
<td>🅴️</td>
<td>🅵️</td>
<td>🅶️</td>
<td>🅷️</td>
<td>🅸️</td>
<td>🅹️</td>
<td>🅺️</td>
<td>🅻️</td>
<td>🅼️</td>
<td>🅽️</td>
</tr>
<tr>
<td>1011</td>
<td>🅱️</td>
<td>🅲️</td>
<td>🅳️</td>
<td>🅴️</td>
<td>🅵️</td>
<td>🅶️</td>
<td>🅷️</td>
<td>🅸️</td>
<td>🅹️</td>
<td>🅺️</td>
<td>🅻️</td>
<td>🅼️</td>
<td>🅽️</td>
</tr>
<tr>
<td>1100</td>
<td>🅱️</td>
<td>🅲️</td>
<td>🅳️</td>
<td>🅴️</td>
<td>🅵️</td>
<td>🅶️</td>
<td>🅷️</td>
<td>🅸️</td>
<td>🅹️</td>
<td>🅺️</td>
<td>🅻️</td>
<td>🅼️</td>
<td>🅽️</td>
</tr>
<tr>
<td>1101</td>
<td>🅱️</td>
<td>🅲️</td>
<td>🅳️</td>
<td>🅴️</td>
<td>🅵️</td>
<td>🅶️</td>
<td>🅷️</td>
<td>🅸️</td>
<td>🅹️</td>
<td>🅺️</td>
<td>🅻️</td>
<td>🅼️</td>
<td>🅽️</td>
</tr>
<tr>
<td>1110</td>
<td>🅱️</td>
<td>🅲️</td>
<td>🅳️</td>
<td>🅴️</td>
<td>🅵️</td>
<td>🅶️</td>
<td>🅷️</td>
<td>🅸️</td>
<td>🅹️</td>
<td>🅺️</td>
<td>🅻️</td>
<td>🅼️</td>
<td>🅽️</td>
</tr>
<tr>
<td>1111</td>
<td>🅱️</td>
<td>🅲️</td>
<td>🅳️</td>
<td>🅴️</td>
<td>🅵️</td>
<td>🅶️</td>
<td>🅷️</td>
<td>🅸️</td>
<td>🅹️</td>
<td>🅺️</td>
<td>🅻️</td>
<td>🅼️</td>
<td>🅽️</td>
</tr>
</tbody>
</table>
Operating Instructions

Examples of 8-Bit and 4-Bit Data Transfer Operation

Display Initialization
Each time the module is turned on or reset, an initialization procedure must be executed. The procedure consists of sending a sequence of hex codes from the microprocessor or parallel I/O port. The initialization sequence turns on the curser, clears the display, and sets the module onto an auto-increment mode.

The initial hex code 30, 34, or 38 is sent two or more times to ensure the module enters the 8-bit or 4-bit data mode. All the initialization sequences are performed under the condition of Register Select (RS) = 0 (low) and Read/Write (RW) = 0 (low).

The 4-bit data bus microcontroller may operate the display module by sending the initialization sequence in 4-bit format. Since 4-bit operation requires the data to be sent twice over the higher 4-bit bus lines (D4-D7), memory requirements are doubled.

A. Example for the Module with 5 x 7 Character Format Under 8-Bit Data Transfer

<table>
<thead>
<tr>
<th>Hex Code</th>
<th>D0 D1 D2 D3 D4 D5 D6</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) 38 (Hex)</td>
<td>0 0 1 1 1 0 0</td>
<td>Function Set → 5-bit Data Length + 2 Line + 5 x 7 Dot Format</td>
</tr>
<tr>
<td>(2) 06 (Hex)</td>
<td>0 0 0 0 0 1 1</td>
<td>Entry Mode Set → Increment one + No Shift</td>
</tr>
<tr>
<td>(3) 06 (Hex)</td>
<td>0 0 0 0 0 1 1</td>
<td>Display ON/OFF Control + Cluster ON + Blink OFF</td>
</tr>
<tr>
<td>(4) 80 (Hex)</td>
<td>0 0 0 0 0 0 1</td>
<td>Display Clear + DD 'RAM Address Set + 1st Digit</td>
</tr>
</tbody>
</table>

Note:
1) Both RS and RW terminals shall be '0' in this sequence.
2) RS, RW and Data are latched at the falling edge of the Enable signal, (falling edge is typically 10ns/Sec; Max. 20ns/Sec).
3) L4044 has I obe initialized on E1 and E2 respectively.
**OPERATING INSTRUCTIONS**

**EXAMPLES OF 8-BIT AND 4-BIT DATA TRANSFER OPERATION**

B. EXAMPLE FOR THE MODULE WITH 5 x 7 CHARACTER FORMAT UNDER 4-BIT DATA TRANSFER

![Flow Chart](image)

<table>
<thead>
<tr>
<th>Hex Code</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Hex)</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 (Hex)</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>2 (Hex)</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>3 (Hex)</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>4 (Hex)</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>5 (Hex)</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>6 (Hex)</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>7 (Hex)</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

**Note:**
1) Both RS and RW terminals shall be "0" in this sequence.
2) RS, R/W, and Data are latched at the falling edge of the Enable signal.
3) Enable signal has to be sent after every 4-bit Data transfer.
Lampiran 4 : Gambar ISP (In-System Programming) kabel
Lampiran 6:
Program List Kunci Elektronik
Program pengendali pintu dengan kode PIN
menggunakan keypad 4x4 dan LCD 16x2 mode data 8-bit
Bahasa pemrograman = bahasa C, Compiler = Small Device C Compiler

#include <cat89x51.h>  //header jenis mikrokontroler at89x51

#define LCD_data P0  //inisialisasi nama (alias) register
#define LCD_LMP  P1_0
#define LCD_RW  P2_1
#define LCD_RW  P2_2
#define LCD_E  P2_3

#define row1  P3_0
#define row2  P3_1
#define row3  P3_2
#define row4  P3_3
#define col1  P3_4
#define col2  P3_5
#define col3  P3_6
#define col4  P3_7

#define buffer  P2_4
#define close_door  P1_5
#define open_door  P2_6

#define limit_open  P1_1
#define limit_close  P1_2
#define led  P1_3

bit at 0x20  pilih;  //alosasi register RAM bitit at 0x21  batal;
bit at 0x22  pin;
bit at 0x23  pukr;
bit at 0x24  buzz;

data unsigned char node,nilai,pojal,cahaj, //alosasi register RAM bit
dig0,dig1,dig2,dig3,dig4,dig5,dig6,dig7;

code unsigned char tamp0[16] = "*";  //alosasi memori

code unsigned char tamp1[16] = "*";  //pada memori FLASH

code unsigned char tamp2[16] = "*";  //* SE"

code unsigned char tamp3[16] = "*";  //* SELA"

code unsigned char tamp4[16] = "*";  //* SELAMAT"

code unsigned char tamp5[16] = "*";  //* SELAMAT"

code unsigned char tamp6[16] = "*";  //* SELAMAT"

code unsigned char tamp7[16] = "*";  //* SELAMAT"

code unsigned char tamp8[16] = "*";  //* SELAMAT"

code unsigned char tamp9[16] = "*";  //* SELAMAT"

code unsigned char tamp10[16] = "*";  //* SELAMAT"

code unsigned char tamp11[16] = "*";  //* SELAMAT"

code unsigned char tamp12[16] = "*";  //* SELAMAT"

code unsigned char tamp13[16] = "*";  //* SELAMAT"

code unsigned char tamp14[16] = "*";  //* SELAMAT"

code unsigned char tamp15[16] = "*";  //* SELAMAT"

code unsigned char tamp16[16] = "*";  //* SELAMAT"

code unsigned char tamp17[16] = "*";  //* SELAMAT"

code unsigned char tamp18[16] = "*";  //* SELAMAT"

code unsigned char tamp19[16] = "*";  //* SELAMAT"

code unsigned char tamp20[16] = "*";  //* SELAMAT"
void tunda(unsigned char val) { //sub rutin tundaan
    unsigned char i,j;
    for(i=0;i<255;i++) //tundaan selama 255 x val(variabel)
        if(i%5==0) //j=0,jx=val;j++)
            iklus;
}

void tulis_instr(unsigned char cmd) { //sub rutin menuliskan instruksi pada LCD 16x2
    unsigned int i;
    LCD_Cmd=0x10; //LCD_data=cmd;
    instruksi
    for(i=0;i<10;i++) //pin kendali LCD pada mode tulis
        //lebar pulse handshake kendali LCD
        LCD_E=0;
}

void tulis_data(unsigned char chr) { //sub rutin menuliskan instruksi pada LCD 16x2
    unsigned int i;
    LCD_Cmd=0x10; //LCD_data=chr;
    instruksi
    for(i=0;i<10;i++) //pin kendali LCD pada mode tulis data
        //lebar pulse handshake kendali LCD
        LCD_E=0;
}

void init_LCD() { //sub rutin inisialisasi LCD 16x2
    tulis_instr(0x38); //reset LCD step-1
    tulis_instr(0x38); //reset LCD step-2
    tulis_instr(0x28); //reset LCD step-3
    5x7 dot
    tulis_instr(0x06); //mode LCD: lebar data 8-bit, 2-row dan
    bergerak
    tulis_instr(0x0c); //mode LCD: panah bawah 1 & tidak
    kedip pada
    tulis_instr(0x01); //layar bersih
void beep()
{
    buzzer=1; tunda(4); buzzer=0;
}

void load()   //sub rutil pengoah nilai dari masukan keypad 4x4
{
    if(posisi==0)   //alokasi register untuk digit kode ke-1
    {
        tulis_inst(0xc5);   //tampilkan tanda '*' pada LCD
        dig1=nilai; posisi=posisi+1;   //simpan nilai
    }
    else if(posisi==1)   //alokasi register untuk digit kode ke-2
    {
        tulis_inst(0xc6);   //tampilkan tanda '*' pada LCD
        dig2=nilai; posisi=posisi+1;   //simpan nilai
    }
    else if(posisi==2)   //alokasi register untuk digit kode ke-
    {
        tulis_inst(0xc7);   //tampilkan tanda '*' pada LCD
        dig3=nilai; posisi=posisi+1;   //simpan nilai
    }
    else if(posisi==3)   //alokasi register untuk digit kode ke-
    {
        tulis_inst(0xc8);   //tampilkan tanda '*' pada LCD
        dig4=nilai; posisi=posisi+1;   //simpan nilai
    }

    else
    {
        //simpan nilai
    }
}

void scan_keypad()   //sub rutin scanning keypad 4x4
{
    col1 = 0;   //scanning pada col1
    if(row1==0)
    {
        beep(); nilai=1; load(); while(row1==0){}   //angka '1'
    }
    else if(row2==0)
    {
        beep(); nilai=2; load(); while(row2==0){}   //angka '2'
    }
    else if(row3==0)
    {
        beep(); nilai=3; load(); while(row3==0){}   //angka '3'
    }
    else if(row4==0)
    {
        beep(); buzz=1; while(row4==0){}   //kode '*'
    }

    col2 = 0;   //scanning pada col2
    if(row1==0)
    {
        beep(); nilai=2; load(); while(row1==0){}   //angka '2'
    }
    else if(row2==0)
    {
        beep(); nilai=3; load(); while(row2==0){}   //angka '3'
    }
    else if(row3==0)
    {
        beep(); nilai=4; load(); while(row3==0){}   //angka '4'
    }
    else if(row4==0)
    {
    }
```c
{beep!;nilai=0;load();while(row4==0){}} //angka '0'
else ; }P3 = 0xff;

col3 = 0; //scanning pada col3
if [row1==0] {[beep!;nilai=3;load();while(row1==0){}] //angka '3'
else if [row2==0] {[beep!;nilai=6;load();while(row2==0){}] //angka '6'
else if [row3==0] {[beep!;nilai=9;load();while(row3==0){}] //angka '9'
else if [row4==0] {[beep!;puk=1;while(row4==0){}] //kode '8'
else ; }P3 = 0xff;

col4 = 0; //scanning pada col4
if [row1==0] {[beep!;while(row1==0){}] //bunuh 'PIN'
else if [row2==0] {[beep!;batal=1;while(row2==0){}] //bunuh 'HAPUS'
else if [row3==0] {[beep!;while(row3==0){}] //bunuh 'TUTUP'
else if [row4==0] {[beep!;pilih=1;while(row4==0){}] //bunuh 'SENAR'
else ; }P3 = 0xff;
}

void proses_data() { /sub rutin pengendali masukkan nilai keypad 4x4

while (pilih==0) { /mendeteksi penekanan tombol

'SENAR'

{ scan_keypad(); //sub rutin scanning keypad 4x4
if [batal==1] { /mendeteksi tombol 'HAPUS'

digo=0;dig1=0;dig2=0;dig3=0;dig4=0; //membersihkan register password
posisi=7;batal=0; //mengembalikan pd kondisi
defaul:

tulis_inst(0x00); //menampilkan '0'
for[i=0;i<6;i++]
	2 LCD (tulis_data '' );
} tunda();
}

} // -----------

void buka_pintu() { //sub rutin membuka pintu

close_door=0;open_door=1;led=0; //apakah pengendali pintu aktif membuka
dan led menyal
while [limit_open==1]{}
close_door=0;open_door=0;led=1; //jika ya, motor pengendali pintu
dipadukan dan led pad
}

void tutup_pintu() { //sub rutin menutup pintu

close_door=1;open_door=0;led=0; //motor pengendali pintu aktif menutup
dan led menyal
while [limit_close==1]{}
close_door=0;open_door=0;led=1; //jika ya, motor pengendali pintu
dipadukan dan led pad
```
void tampil_bersor()
{
    unsigned char i;
    tulus_inst(0x80);
    for(i=0;i<16;i++)
    {
        tulus_data(tampil0[i]); //tampilkan frame ke-1
        tunda(100);
        tulus_inst(0x80);
        for(i=0;i<16;i++)
        {
            tulus_data(tampil1[i]); //tampilkan frame ke-2
            tunda(100);
            tulus_inst(0x80);
            for(i=0;i<16;i++)
            {
                tulus_data(tampil2[i]); //tampilkan frame ke-3
                tunda(100);
                tulus_inst(0x80);
            }
            tulus_data(tampil3[i]); //tampilkan frame ke-4
            tunda(100);
        }
        tulus_data(tampil4[i]); //tampilkan frame ke-5
        tunda(100);
        tulus_inst(0x80);
        for(i=0;i<16;i++)
        {
            tulus_data(tampil5[i]); //tampilkan frame ke-6
            tunda(100);
            tulus_inst(0x80);
            for(i=0;i<16;i++)
            {
                tulus_data(tampil6[i]); //tampilkan frame ke-7
                tunda(100);
                tulus_inst(0x80);
            }
            tulus_data(tampil7[i]); //tampilkan frame ke-8
            tunda(100);
        }
        tulus_data(tampil8[i]); //tampilkan frame ke-9
        tunda(100);
        tulus_inst(0x80);
        for(i=0;i<16;i++)
        {
            tulus_data(tampil9[i]); //tampilkan frame ke-10
            tunda(100);
            tulus_inst(0x80);
            for(i=0;i<16;i++)
            {
                tulus_data(tampil10[i]); //tampilkan frame ke-11
                tunda(100);
                tulus_inst(0x80);
                for(i=0;i<16;i++)
                {
                    tulus_data(tampil11[i]); //tampilkan frame ke-12
                    tunda(100);
                    tulus_inst(0x80);
                }
                tulus_data(tampil12[i]); //tampilkan frame ke-13
                tunda(100);
                tulus_inst(0x80);
            }
            tulus_data(tampil13[i]); //tampilkan frame ke-14
            tunda(100);
        }
        tulus_inst(0x80);
    }
}
for(i=0; i<16; i++)
{tulis_data(tampil4[i]);} //tampilkan frame ke-15

tunda(100);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil5[i]);} //tampilkan frame ke-16

tunda(100);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil6[i]);} //tampilkan frame ke-17

tunda(200);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil7[i]);} //tampilkan frame ke-18

tunda(100);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil8[i]);} //tampilkan frame ke-19

tunda(100);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil9[i]);} //tampilkan frame ke-20

tunda(100);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil10[i]);} //tampilkan frame ke-21

tunda(100);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil11[i]);} //tampilkan frame ke-22

tunda(100);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil12[i]);} //tampilkan frame ke-23

tunda(100);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil13[i]);} //tampilkan frame ke-24

tunda(100);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil14[i]);} //tampilkan frame ke-25

tunda(100);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil15[i]);} //tampilkan frame ke-26

tunda(100);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil16[i]);} //tampilkan frame ke-27

tunda(100);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil17[i]);} //tampilkan frame ke-28

tunda(100);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil18[i]);} //tampilkan frame ke-29

tunda(100);
tulis_inst(0x80);
for(i=0; i<16; i++)
{tulis_data(tampil19[i]);} //tampilkan frame ke-30

...
tunda(100);

tulis_inst(0x80);
for(i=0;j<16;i++)
{tulis_data(tampil30[i])); //tampilkan frame ke-31
}

.dylib main()
{ //program utama pengendali sistem

    unsigned char i;
P2=0x01;mode=1; //pengkondisian awal
    init_LCD(); //melakukan inisialisasi LCD 16x2
    while(1)
    {
        if(mode==1)
        {
            caesar(0);buzzer=0;led=1; //menampilkan kondisi register fungsi
            menuju awal (0)
            while(pin==0)
            {
                tulis_inst(0x01);tunda(8); //bersihkan layar
tampil_geser(); //menampilkan tampilan bergeger
            'SELAMAT DATANG'
            tulis_inst(0x80);
            for(i=0;j<16;i++)
            {tulis_data(info0[i]); //mengambil 16 karakter tampilan
                tulis_inst(0x80);
                for(i=0;i<16;i++)
                {tulis_data(info2[i]); //mengambil 16 karakter tampilan
                    for(i=0;i<10;i++)
                    {co14=0;
                        if(zow1==1){pin=1;}
                        co14=1;
                        tunda(100); //mengubah dari mode 1 menuju mode 2
                    }
                    mode=2;pin=0;
                }
                if(mode==2)
                {
                    tulis_inst(0x80);tunda(8);
                    tulis_inst(0x80);
                    for(i=0;i<16;i++)
                    {tulis_data(info3[i]); //mengambil 16 karakter tampilan
                        proses_data(); //mengambil nilai dari masukan
                    }
                    keypad_cari;
posisi=0;pin=0;
                    if(dig0==0 && dig1==3 && dig2==9 && dig3==7 && dig4==7) //jika kode benar?? maka...
                    {
                        dig0=0;dig1=0;dig2=0;dig3=0;dig4=0;
                        tulis_inst(0x01);tunda(8); //bersihkan layar
tulis_inst(0x80);
                        for(i=0;i<16;i++)
                        {tulis_data(info4[i]); //mengambil 16 karakter tampilan
                            for(i=0;i<13;i++)tunda(255); //mempunyai informasi 'PIN BENAR!
                        }
luka_pintu(); //sub rutin membuka pintu
                    }
                }
            }
        }
    }
tulis_inst(0x80);
for(i=0;i<16;i++)
{tulis_data(info6[i]);}
//mengambil 16 karakter tampilan

for(i=0;i<16;i++)
{tulis_data(info7[i]);}
//mengambil 16 karakter tampilan

P3=0xff;col4=0;
while(row3==1){
col4=1;P3=0xff;
tutup_pintu();
//sub rutin menutup pintu

if(mode==1)
{manuju mode 1

dig0=0;dig1=0;dig2=0;dig3=0;dig4=0;
mode=0;
cacaah=cacaah+1;
tulis_inst(0x01);tunda(0);
tulis_inst(0x80);
for(i=0;i<16;i++)
{tulis_data(info5[i]);}
//mengambil 16 karakter tampilan
for(i=0;i<3;i++)
tunda(255);
if(cacaah==3)
{nysis}

while(row4==1)
{
col1=0;
if(row4==1)
{ //mendeteksi penekanan tombol ""}
else if(row4==0) (buzzer=0;)
//jika ditekan, buzzer padam

col3=0;buzzer=0;
//jika tombol "" ditekan, buzzer ikut padam

for(i=0;i<16;i++)
{tulis_data(info6[i]);}
//mengambil 16 karakter tampilan
for(i=0;i<3;i++)
tunda(255);

proses_data();
keypad 4x4

posisi=0;pin1=0;
if(dig0==9 && dig1==9 && dig2==9 && dig3==9 && dig4==0) //jika
kode benar??? maka...
{
dig0=0;dig1=0;dig2=0;dig3=0;dig4=0;
tulis_inst(0x01);tunda(8); //bersihkan layar
tulis_inst(0x80);
for(i=0;i<16;i++)
[tulis_data(info[4][i]]); //menampilkan informasi 'PIN
BENAR'
for(i=0;i<3;i++)
tunda(255);i}
mode=1; //mengubah dari mode 2 kembali
menuju mode 1
}
else
{
dig0=0;dig1=0;dig2=0;dig3=0;dig4=0;
tulis_inst(0x01);tunda(8); //bersihkan layar
tulis_inst(0x80);
for(i=0;i<16;i++)
[tulis_data(info[5][i]]); //menampilkan informasi 'PIN SALAH'
for(i=0;i<3;i++)
tunda(255);i}
while(1)
{ //terjebak di sini!!!
LCD_lmp=1;led=0;tunda(255); //dengan led & lampu LCD berkedip-
keempat
LCD_lmp=0;led=1;tunda(255);
}
Lampiran 7: Gambar Kunci Elektronik

Gambar 1. Kunci Elektronik Tampak Depan

Gambar 2. Kunci Elektronik Tampak Samping
Gambar 3. Membuka Pintu

Gambar 4. Menutup Pintu