Contents

No$x51 Features/About

Heya
If anybody uses this program, or otherwise treats it to be useful, please let me know - I'd be happy about some feedback :-)
If using the program for commerical purposes please contact me (see also chapter about External Hardware for possible support/improvements).
The program is free for non-commerical use (anyways, small donations would be highly welcome).

http://problemkaputt.de/x51.htm - no$x51 homepage
http://problemkaputt.de/email.htm - email address (spam shielded)

http://problemkaputt.de/address.htm - my mailing address

Features
This program is written in tight 80X86 assembler code, and covers the basic nocash emulation/debugging features:

* 8051/8031/P8xCE558 Emulator
* Debugger/Disassembler with Symbolic Information (Labels)
* Assembler (Single-Line Input, and Source Code Assembler)
* (Dis-)Assembler supports native 8051 and friendly Z80/80X86 syntax
* Warning Messages on bad I/O and stack overflows, etc.
* FEEPROM Upload Function (Serial Boot)
* Emulates External LCD 16x2 Charcters
* Emulates External Numeric Keypad
* DOS version for PC/XT 8086 with 80x25 MDA and up
* Windows version for 80386SX / Windows 95 and worse

The current version emulates the CPU including internal SFR registers such like timers and prescalers - it still lacks emulation of (or redirection to) external hardware.

Memory and Register Map

Internal RAM (MOV,ADD,PUSH,etc.)
```
  00-07  R0..R7  ;registers R0..R7 (bank 0, default)
                 ;<--- initially SP=07 (stack incrementing at 08 and up).
  08-0F  R0..R7  ;registers R0..R7 (bank 1) or normal RAM
  10-17  R0..R7  ;registers R0..R7 (bank 2) or normal RAM
  18-1F  R0..R7  ;registers R0..R7 (bank 3) or normal RAM
  20-2F          ;bit-addressable RAM (16x8 bits) or normal RAM
  30-7F          ;normal RAM
  80-FF          ;558 only - extra internal RAM - addressable by @Ri & SP only
```

External RAM (MOVX only)
```
  0000-00FF  AUX RAM
  0100-01FF  AUX RAM
  0200-02FF  AUX RAM
  ...
  8000..
```

Code ROM/EEPROM (MOVC only)

```
  0000-7FFF  ;internal EEPROM (P89CE558 only)
  8000-FBFF  ;external memory / user space
  FC00-FFFF  ;internal BIOS ROM (all 558 only ?)
```

Internal EEPROM and BIOS can be disabled, allowing to use the whole 64Kbytes address space for external memory. Data can be written into the FEEPROM from inside of the user program by using BIOS functions.
Address 0000h is the reset vector (should contain a single JMP instruction), addresses 0003h, 000Bh, 0013h, 001Bh, ..., 0073h are interrupt vectors.

SFR (Special Function Registers)

```
  80  P0         A0  P2         C0* P4         E0  A/ACC
  81  SP         A1  -          C1  -          E1  -
  82  DPL        A2  -          C2  -          E2  -
  83  DPH        A3  -          C3  -          E3  -
  84  -          A4  -          C4  -          E4  -
  85  -          A5  -          C5  -          E5  -
  86* ADRSL0     A6* ADRSL2     C6* ADRSL4     E6* ADRSL6
  87  PCON       A7  -          C7* P5         E7* ADPSS
  88  TCON       A8  IEN0/IEC   C8* TM2IR      E8* IEN1
  89  TMOD       A9* CML0       C9* CMH0       E9  -
  8A  TL0        AA* CML1       CA* CMH1       EA* TM2CON
  8B  TL1        AB* CML2       CB* CMH2       EB* CTCON
  8C  TH0        AC* CTL0       CC* CTH0       EC* TML2
  8D  TH1        AD* CTL1       CD* CTH1       ED* TMH2
  8E  -          AE* CTL2       CE* CTH2       EE* STE
  8F  -          AF* CTL3       CF* CTH3       EF* RTE
  90  P1         B0  P3         D0  PSW        F0  B
  91  -          B1  -          D1  -          F1  -
  92  -          B2  -          D2  -          F2  -
  93  -          B3  -          D3  -          F3  -
  94  -          B4  -          D4  -          F4  -
  95  -          B5  -          D5  -          F5  -
  96* ADRSL1     B6* ADRSL3     D6* ADRSL5     F6* ADRSL7
  97  -          B7  -          D7* ADCON      F7* ADRSH
  98  S0CON/SCON B8  IP0/IPC    D8* S1CON      F8* IP1
  99  S0BUF/SBUF B9  -          D9* S1STA      F9* PLLCON
  9A  -          BA  -          DA* S1DAT      FA* XRAMP
  9B  -          BB  -          DB* S1ADR      FB**FMCON
  9C  -          BC  -          DC  -          FC* PWM0
  9D  -          BD  -          DD  -          FD* PWM1
  9E  -          BE  -          DE  -          FE* PWMP
  9F  -          BF  -          DF  -          FF* T3
```

Notes:

```
  *   P8xCE558 only (not 8031/8051)
  **  P89CE558 only (not 8031/8051/P80CE558/P83CE558)
  IEN0,S0CON,S0BUF,IP0 are new 558-expressions for original IEC,SCON,SBUF,IPC.
  Accumulator may be called A or ACC. In source, ACC forces a "direct" operand.
  Registers at SFR addresses n*8 are bit-addressable (eg. P0,TCON,P1,etc).
  DPL,DPH are lower/upper bits of DPTR.
```

Bit-Addressable SFR Registers

Register	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
F8 IP1	PT2	PCM2	PCM1	PCM0	PCT3	PCT2	PCT1	PCT0
F0 B	B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0
E8 IEN1	ET2	ECM2	ECM1	ECM0	ECT3	ECT2	ECT1	ECT0
E0 A	A.7	A.6	A.5	A.4	A.3	A.2	A.1	A.0
D8 S1CON	CR2	ENS1	STA	STO	SI	AA	CR1	CR0
D0 PSW	CY	AC	F0	RS1	RS0	OV	F1	P
C8 TM2IR	T2OV	CMI2	CMI1	CMI0	CTI3	CTI2	CTI1	CTI0
C0 P4	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
B8 IP0	--	PAD	PS1	PS0	PT1	PX1	PT0	PX0
B0 P3	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
A8 IEN0	EA	EAD	ES1	ES0	ET1	EX1	ET0	EX0
A0 P2	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
98 S0CON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
90 P1	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
88 TCON	TF1	TR1	ZF0	TR0	IE1	IT1	IE0	IT0
80 P0	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0

Note that normal Set/Reset operations may be performed for non-bit-addressable registers by "ANL/ORL <direct>,#Imm" either.

External I/O Ports

SFRs : P0, P1, P2, P3, P4, P5

80h - P0 - 8bit Open Drain Bidirectional I/O Port (Read/Write/Bit-adressable)
90h - P1 - 8bit Quasi-Bidirectional I/O Port (Read/Write/Bit-adressable)
A0h - P2 - 8bit Quasi-Bidirectional I/O Port

```
  with internal pull-ups (Read/Write/Bit-adressable)
```

B0h - P3 - 8bit Quasi-Bidirectional I/O Port (Read/Write/Bit-adressable)
C0h - P4 - 8bit Quasi-Bidirectional I/O Port (Read/Write/Bit-adressable)
C7h - P5 - 8bit Input Port (Read Only)

Timers

Timer 0 and 1

Additionally, a seconds timer is available (P8xCE558 with 32kHz oscillator only), for details see PLLCON description in SYS chapter.

Timer 0 and 1

SFRs : TCON, TMOD, TL0, TH0, TL1, TH1 (and P3.4-5)

8Ah/8Ch - TL0/TH0 - T0 - Timer 0 Counter (Read/Write)
8Bh/8Dh - TL1/TH1 - T1 - Timer 1 Counter (Read/Write)
16bit timer/counter registers for each timer 0 and 1. May be split into 8bit timer/counters with associated reload value or prescaler, depending on timer mode.
When reading a 16bit value, either temporarily stop the timer, or first read MSB then LSB then re-read MSB - and retry if it has changed in the meantime.

88h - TCON - Timer/Counter Control Register (Read/Write, Bit-Addressable)
```
  Bit  Name     Expl.
  0,2  IT0,IT1  Interrupt 0,1 Type Control  (0=Low,  1=Falling edge)
  1,3  IE0,IE1  Interrupt 0,1 Edge Flag     (0=None, 1=IRQ)
  4,6  TR0,TR1  Timer 0,1 Run Control       (0=Stop, 1=Run)
  5,7  TF0,TF1  Timer 0,1 Overflow Flag     (0=None, 1=IRQ)
```
The four IRQ-bits are automatically set/cleared by hardware when sensing/excuting an interrupt. The other four bits are controlled by software only.

89h - TMOD - Timer/Counter Control Register (Read/Write)
```
  Bit  Name  Expl.
  0-1  M0-1  Timer 0 Mode            (0-3, see below)
  2    C/T   Timer 0 Selector        (0=Timer, 1=Counter)
  3    GATE  Timer 0 Gating Control  (0=Normal, 1=Stop Timer while /INT0=LOW)
  4-5  M0-1  Timer 1 Mode            (0-2, see below, 3=Timer stopped)
  6    C/T   Timer 1 Selector        (0=Timer, 1=Counter)
  7    GATE  Timer 1 Gating Control  (0=Normal, 1=Stop Timer while /INT1=LOW)
```
Timer 0 Modes (for Timer 1 Modes use TH1/TL1 respectively)
```
  0   8bit Timer/Counter TH0, each with 5bit prescaler TL0 (8048 Mode).
  1   16bit Timer/Counter, TH0 and TL0 are cascaded.
  2   8bit auto-reload Timer/Counter, TL0=timer/counter, TH0=reload value.
  3   8bit TL0 Timer/Counter 0, plus 8bit TH0 Timer 1 (Timer 0 only).
```
Timer/Counter Selector
```
  Timer - Incremented at fCLK/12
  Counter - Incremented on Falling Edge of external input
```
The counter 0/1 input pins T0/T1 (P3.4/P3.5) may be pulsed at 0Hz through max fClk/24.

Timer 2

SFRs : TM2IR, TM2CON, TML2, TMH2 (and P1.4-5) - (Raw Timer)
SFRs : CTCON, CTL0-3, CTH0-3 (and P1.0-3) - (Timer/Capture/Ext.Interrupt)
SFRs : STE, RTE, CML0-2, CMH0-2 (and P4.0-7) - (Timer/Compare)

ECh/EDh - TML2/TMH2 - T2 - 16bit Timer 2 Counter (Read Only)
16bit readonly (!) timer/counter register. The register is not loadable, however, when enabled in T2ER (TM2CON.5), the timer may be reset by 0-to-1 transition of RT2 (P1.5); which'd be normally generated external hardware (but could be generated by software output to P1.5 either - provided that external hardware is not dragging that pin to low).

When reading the 16bit value, either temporarily stop the timer, or first read MSB then LSB then re-read MSB - and retry if it has changed in the meantime.

EAh - TM2CON - Timer 2 Control Register (Read/Write)
```
  Bit  Name     Expl.
  0-1  T2MS0-1  Timer 2 Mode Select (0=Halted, 1=Timer, 2=Reserved, 3=Counter)
  2-3  T2P0-1   Timer 2 Prescaler   (0-3=Divide clock source by 1,2,4,8)
  4    T2BO     Timer 2 Byte Overflow Interrupt Flag (0=None, 1=IRQ)
  5    T2ER     Timer 2 External Reset Enable (0=Disable, 1=Enable)
                (When enabled, T2 becomes reset on Raising Edge of RT2/P1.5)
  6    T2IS0    Timer 2 Byte Overflow Interrupt Select
  7    T2IS1    Timer 2 16-bit Overflow Interrupt Select
```
Both Byte and 16bit Overflows are sharing the same interrupt vector, either one or both may be enabled, 16bit Overflow flag located in TM2IR.

C8h - TM2IR - Timer 2 Interrupt Flag Register (Read/Write, Bit-addressable)
```
  Bit  Name     Expl.
  0-3  CTI0-3   CT0-3 (Capture) Interrupt Flags (0=None, 1=IRQ)
  4-6  CMI0-2   CM0-2 (Compare) Interrupt Flags (0=None, 1=IRQ)
  7    T2OV     Timer 2 16-bit Overflow Flag    (0=None, 1=IRQ)
```
Note: Additionally, a 8-bit Overflow Flag is located in TM2CON.4 (T2BO).
All Timer 2 interrupt flags must be reset by software.

E8h - CTCON - Capture Control Regtister (Read/Write, Bit-addressable)

```
  Bit      Name    Expl.
  0,2,4,6  CTP0-3  Capture Register 0-3 triggered by falling edge on CT0I-CT3I
  1,3,5,7  CTN0-3  Capture Register 0-3 triggered by raising edge on CT0I-CT3I
```

When triggered, current Timer 2 value is loaded into selected capture register (see CT0-3), and a "capture" interrupt is requested (see TM2IR and IEN1). May be triggered on both raising and/or falling edge, when deselcting both edges, the interrupt input is disabled. When ignoring the captured value, the 'capture' interrupts 0-3 may be treated as 'normal' external interrupts 2-5.

ACh/CCh - CTL0/CTH0 - CT0 - 16bit Capture Register 0 (Read Only)
ADh/CDh - CTL1/CTH1 - CT1 - 16bit Capture Register 1 (Read Only)
AEh/CEh - CTL2/CTH2 - CT2 - 16bit Capture Register 2 (Read Only)
AFh/CFh - CTL3/CTH3 - CT3 - 16bit Capture Register 3 (Read Only)

The 16bit Timer 2 value can be automatically loaded (captured) into any of the CM0-3 registers upon either raising and/or falling edge of any of the CT0I-CT3I Pins, see CTCON Register.

EEh - STE - Compare Set Enable Register (Read/Write)

```
  Bit  Name     Expl.
  0-5  SP40-45  New state for P4.0-5 upon CM0=T2   (0=Don't change, 1=Set)
  6-7  TG46-47  New state for P4.6-7 upon toggle   (0=Set, 1=Reset)
```

Port 4 can be read and written by software without affecting the toggle, set, and reset signals.

EFh - RTE - Compare Reset/Toggle Enable Register (Read/Write)

```
  Bit  Name     Expl.
  0-5  RP40-45  New state for P4.0-5 upon CM1=T2   (0=Don't change, 1=Reset)
  6-7  TP46-47  New state for P4.6-7 upon CM2=T2   (0=Don't change, 1=Toggle)
```

A9h/C9h - CML0/CMH0 - CM0 - Compare Register 0 (Read/Write) - Set
AAh/CAh - CML1/CMH1 - CM1 - Compare Register 1 (Read/Write) - Reset
ABh/CBh - CML2/CMH2 - CM2 - Compare Register 2 (Read/Write) - Toggle

The 16bit values in each CM0-2 are continously compared with the T2 counter value. Matches for CM0-2 may set/reset/toggle outputs of P4.0-7 Pins:

```
  Register  Action   Pins     See also
  CM0       Set      P4.0-5   STE Register
  CM1       Reset    P4.0-5   RTE Register
  CM2       Toggle   P4.6,7   STE and RTE Registers
```

If CM0 and CM1 match at the same time, CM1 will have priority (Reset).
Additionally, when a match occurs, a "compare" interrupt is requested (see TM2IR and IEN1).

Alternative Functions of Port 1 and Port 4 with Timer 2

```
  Port/Pin Alternative Function
  P1.0-3   CT0I-CT3I   External Interrupt Inputs 2-5 (with Timer 2 Capture 0-3)
  P1.4     T2          T2 event input (counter mode), rising edge triggered
  P1.5     RT2         T2 reset input, rising edge triggered
  P1.6-7   -           No special function
  P4.0-5   CMSR0-5     Compare and Set/Reset outputs on T2 match
  P4.6-7   CMT0-1      Compare and Toggle outputs on T2 match
```

Alias name for CT0I-CT3I would be INT2-INT5 (used when ignoring the capture function that is associated to the interrupt input).

Timer 3 (Watchdog)

SFRs : T3 (and PCON.4)

FFh - T3 - Watchdog Timer (Read/Write)
If the /EW Pin is LOW, this 8bit timer register is incremented once every 12*2048 oscillator periods (ie. each 2048 cycles). In case that the timer overflows, the CPU will be reset, and a reset signal will be output at RSTOUT, the software must thus permanently reload the T3 timer in order to keep itself operating.
Even though this automatic reset function is theoretically a rather dangerous feature, it becomes useful especially if the microprocessor is not guarded by human operators, as it allows the system to recover itself even if the software has locked up either because of a programming bug or hardware malfunction.
First write "1" to the WLE Bit (PCON.4), this enables writing to T3.
Then write the reload value to T3, this will reset both the WLE Bit and the 2048-steps-prescaler, and (if WLE was set, and if /EW was LOW), the reload value will be applied in T3, and the software may continue for the specified amount of time.
The Watchdog timeout ranges from 1.5ms (reload FFh) through 0.375s (reload 00h) when using a system clock of 16MHz - or longer, when using a slower system clock.

The /EW Pin
When the /EW Pin is HIGH, the Watchdog function will be activated, and the software MUST reload T3 repeatedly, both the Power Down Mode and the Serial FEEPROM Programming Function cannot be used.
When the /EW Pin is LOW, the Watchdog is disabled, writing to T3 will be rejected, the T3 Timer will be halted, and the Power Down Mode will be available.
The state of the /EW Pin can be read-out at LOADEN (PCON.1) ???

Serial UART/RS232 Port

SFRs : S0CON,S0BUF alias SCON,SBUF (and PCON.7 and Timer)

Full duplex serial I/O port - it can transmit and receive simultaneosly.
Received data may be read out by software even when the hardware already receives a second byte, however, one byte will be lost if the software failed to read-out 1st data by the time when reception of the 2nd data completes.

98h - S0CON/SCON - UART Serial Control Register (Read/Write, Bit-addressable)
The IRQ flags are set by hardware, and must be cleared by software. In mode 0, set at the end of the 8th bit. In other modes, set at the beginning (TI), or in the middle (RI), of the stopbit.

```
  Bit  Name    Expl.
  0    RI      Receive Interrupt Flag    (0=None, 1=IRQ)
  1    TI      Transmit Interrupt Flag   (0=None, 1=IRQ)
  2    RB8     9th received bit (Mode0=Not used, Mode1=Stopbit, Mode2-3=Bit8)
  3    TB8     9th transmit bit (Bit8, set by software, used in Mode2-3 only)
  4    REN     Serial Reception Enable   (0=Disable, 1=Enable)
  5    SM2     Receive Interrupt Mode    (0=Normal, 1=See below)
  6    SM1     SM1=LSB (!) of Serial Mode, see below
  7    SM0     SM0=MSB (!) of Serial Mode, see below
```

Serial Modes, and meaning of SM2=1:

```
  Mode   Expl.              Baudrate        ;When SM2=1, set RI only if...
  0/00h  8-bit Shift register  fCLK/12         ;-[Reserved, SM2 should be 0]
  1/40h  8-bit UART            variable        ;-only if received Stopbit valid
  2/80h  9-bit UART            fCLK/64 or /32  ;-only if received 9th bit RB8=1
  3/C0h  9-bit UART            variable        ;-only if received 9th bit RB8=1
```

Mode 2-3 with SM2=1 are somewhat intended for Multiprocessor systems.

99h - S0BUF/SBUF - UART Serial Transmit Data (Write Only)
99h - S0BUF/SBUF - UART Serial Receive Data (Read Only)

Two separate 8bit registers, one for receive, one for transmit, that are sharing the same SFR address. In all modes, data is transferred LSB first (shifted to the right), and in modes 2-3 a 9th data bit is transmit/received from the S0CON register. This 9th bit may be used as Parity bit - which must be manually produced or verified by software (if so, note that the CPU provides a Parity bit in the PSW register).
Transmit is initiated by writing to S0BUF.

UART Transfer Start

Reception is initiated in Mode 0 when R1=0 and REN=1.
Reception is initiated in Mode 1-3 by incoming Startbit when REN=1.
Transmit is initiated in all modes by writing to S0BUF.

UART Transfer Notes

Mode 0 uses RXD as data line (for both transmit and receive), and outputs the shift clock to TXD, writing to S0CON should be avoided during Mode 0 transmission to avoid spikes on RXD/TXD.
Mode 1-3 are using RXD as Receive data line, and TXD as transmit data line. A transfer consists of one startbit (0), eight or nine data bits (LSB first), and one stopbit (1).

UART - See also IEN0 and Timer.

```
  mov  tmod,#00100001b  ;\  init timer-1 for auto-reload at 32x2400Hz
  mov  th1,#0f4h        ; > ("for used as gated 16-bit counter" ???)
  setb tr1              ;/
```

In modes 1-3, the Baudrate may be doubled by setting SMOD in PCON.7.

Serial I2C-Bus Port

SFRs : S1CON, S1STA, S1DAT, S1ADR

D8h - S1CON - I2C-bus Control Register (Read/Write, Bit-addressable)

```
  Bit  Name    Expl.
  0-1  CR0-1   Clock Rate LSBs (MSB see below), depending on System Clock:
         Clock   0     1     2     3     4     5     6     7     CR0-2 value
       __Divider_60____1600__40____30____240___3200__160___120___osc.periods___
         12MHz   200   7.5   300   400   50    3.75  75    100   kHz (kbit/sec)
         16MHz   266.7 10    400   -     66.7  5     100   -     kHz (kbit/sec)
        Values higher than 100kHz for "fast-mode" I2C-bus applications only,
        not compatible with older I2C-bus systems. CR0-2 used in master mode
        only - slave mode automatically synchronizes to any rate up to 400kHz.
  2    AA      Assert Acknowledge flag  (0=None, 1=Return Acknowledge)
        When enabled, acknowledge is returned when:
        - Own slave address or General call address is received.
        - Data byte is received as master receiver or selected slave receiver.
  3    SI      Serial Interrupt flag  (0=None, 1=IRQ)
        While SI is set, SCL remains LOW and the transfer is suspended.
        SI must be reset by software. IRQs are generated when:
        - A START condition is generated in master mode.
        - Own slave address or general call addr has been received during AA=1.
        - Data byte has been received or transmitted in master mode
          (even if arbitration is lost).
        - Data byte has been received or transmitted as selected slave.
        - STOP or START received as selected slave receiver/transmitter.
  4    STO     STOP Flag        (0=Nope, 1=Stop/Recover)
        In Master mode, issues a STOP condition to the bus. In slave mode,
        recovers from error (without issuing STOP to bus). This flag is
        cleared by hardware when sensing a STOP on the bus, or when ENS1=0.
  5    STA     START Flag       (0=Nope, 1=Generate start condition, see below)
        In Slave mode, start condition is generated once that the bus
        becomes free. In Master mode, condition is repeatedly generated.
  6    ENS1    Serial I/O Enable    (0=Disable,Reset,SDA=SCL=high-Z, 1=Enable)
  7    CR2     Clock Rate MSB, see CR0-1 description above
```

D9h - S1STA - I2C-bus Serial Status Register (Read Only)

The lower 3bits of this register are guaranteed to be always zero, this is explicitely intended for using the S1STA value as jump vector offset in increments of eight.

```
  Bit  Name   Expl.
  0-2  0      All Zeros
  3-7  SC0-4  Status Code (with above zero-bits, 00-F8)
```

S1STA Codes for MST/TRX Mode:

```
  08h  A START condition has been transmitted
  10h  A repeated START condition has been transmitted
  18h  SLA and W have been transmitted, ACK has been received
  20h  SLA and W have been transmitted, /ACK received
  28h  DATA and S1DAT have been transmitted, ACK received
  30h  DATA and S1DAT have been transmitted, /ACK received
  38h  Arbitration lost in SLA, R/W or DATA
```

S1STA Codes for MST/REC Mode:

```
  38h  Arbitration lost while returning /ACK
  40h  SLA and R have been transmitted, ACK received
  48h  SLA and R have been transmitted, /ACK received
  50h  DATA has been received, ACK returned
  58h  DATA has been received, /ACK returned
```

S1STA Codes for SLV/REC Mode:

```
  60h  Own SLA and W have been received, ACK returned
  68h  Arbitration lost in SLA, R/W as MST. Own SLA and W have been
       received, /ACK returned
  70h  General CALL has been received, ACK returned
  78h  Arbitration lost in SLA, R/W as MST. General CALL has been received
  80h  Previously addressed with own SLA. DATA byte received, ACK returned
  88h  Previously addressed with own SLA. DATA byte received, /ACK returned
  90h  Previously addressed with general call. DATA byte has been received,
       ACK has been returned
  99h  Previously addressed with general call. DATA byte has been received,
       /ACK has been returned
  A0h  A STOP condition or repeated START condition has been received while
       still addressed as SLV/REC or SLV/TRX
```

S1STA Codes for SLV/TRX Mode:

```
  A8h  Own SLA and R have been received, ACK returned
  B0h  Arbitration lost in SLA, R/W as MST. Own SLA and R have been
       received, /ACK returned
  B8h  DATA byte has been transmitted, ACK returned
  C0h  DATA byte has been transmitted, /ACK returned
  C8h  Last DATA byte has been transmitted (AA=logic 0), ACK received
```

S1STA Miscellaneous Codes:

```
  00h  Bus error during MST mode or selected SLV mode, due to an erroneous
       START or STOP condition
  F8h  No relevant information available, SI not set
```

Abbreviations used:

```
  MST  Master                 R     Read bit
  SLV  Slave                  W     Write bit
  TRX  Transmitter            ACK   Acknowledgement (acknowledge bit = 0)
  REC  Receiver               /ACK  Not acknowledgement (acknowledge bit = 1)
  SLA  7-bit slave address    DATA  8-data byte to or from I2C-bus
```

DAh - S1DAT - I2C-bus Data Shift Register (Read/Write)

Contains 8bit serial data, Bit 7 is received or transmitted first, ie. data is shifted left.

DBh - S1ADR - I2C-bus Address Register (Read/Write)

```
  Bit  Name    Expl.
  0    GC      General Call address state  (0=Not Recognized, 1=Recognized)
  1-7  SLA0-6  Own Slave Address           (00-7F)
```

The Slave Address determines the address to which the controller will respond when programmed as slave receiver/transmitter.

Analog/Digital Converter

SFRs : ADCON, ADPSS, ADRSL0-7, ADRSH (and P5)

D7h - ADCON - ADC Control Register (Read/Write)

```
  Bit  Name     Expl.
  0    ADSFE    Start A/D conversion on falling edge at ADEXS-Pin (0=No, 1=Yes)
  1    ADSRE    Start A/D conversion on raising edge at ADEXS-Pin (0=No, 1=Yes)
  2    ADCSA    Scan Selected analog inputs          (0=One-Time, 1=Continous)
  3    ADSST    Start and Status              (0=Inactive/Stop, 1=Active/Start)
  4    ADINT    ADC Interrupt on completion of selected inputs  (0=None, 1=IRQ)
  5    ADPOS    Reserved for future use (Always write "0")
  6-7  ADPR0-1  Prescaler Control       (0-3=Divide by 2,4,6,8)
```

Scan is started when ADSST changes from 0 to 1 (either by software, or by ADEXS input), the bit is automatically cleared upon completion of a One-Time scan, in Continous mode it remains set. Clearing by software ADSST will abort the current scan.
Interrupts are requested upon completion of all selected inputs (ie. once in One-Time mode, and repeatedly in Continous mode). ADINT must be cleared by software, but it cannot be set by software.

E7h - ADPSS - A/D Input Port Scan-Select Register (Read/Write)

```
  Bit  Name     Expl.
  0-7  ADPSS0-7  Select analog input 0-7 at P5.0-7   (0=Skip, 1=Select)
```

All selected analog inputs are scanned during the auto-scan loop (starting with lowest selected bit position) (each one once in One-Time mode, or repeatedly in Continous mode), A/D conversion cannot be started when all ADPSS bits are zero. When writing to ADPSS while scan is in progress, changes aren't recognized until the current auto-scan loop ends, changes are then applied for the next loop.

86h - ADRSL0 - A/D Result Register 0 (Read Only)
96h - ADRSL1 - A/D Result Register 1 (Read Only)
A6h - ADRSL2 - A/D Result Register 2 (Read Only)
B6h - ADRSL3 - A/D Result Register 3 (Read Only)
C6h - ADRSL4 - A/D Result Register 4 (Read Only)
D6h - ADRSL5 - A/D Result Register 5 (Read Only)
E6h - ADRSL6 - A/D Result Register 6 (Read Only)

F6h - ADRSL7 - A/D Result Register 7 (Read Only)
Lower 8bits of conversion results for each of the eight analog inputs.
Reading from ADRSL0-7 automatically latches the upper 2bits of the conversion result (which is sized 10bits in total) into the ADRSH register.
The latched value remains unchanged even if an active scan-loop samples new data, thus, when reading ADRSLn and then ADRSH, there is no danger that ADRSH conatins 'newer' data than ADRSLn.

F7h - ADRSH - A/D Result Register High (Read Only)
Contains the upper 2bits from the most recently read ADRSL0-7 conversion result (see above). Bit2-7 of ADRSH are always zero.

Alternate use for P5 Digital Input

Pulse Width Modulated Outputs

SFRs : PWM0, PWM1, PWMP

FCh - PWM0 - Pulse Width Register for /PWM0 (Read/Write)
FDh - PWM1 - Pulse Width Register for /PWM1 (Read/Write)
These registers specify the LOW:HIGH ration for the /PWM0 and /PWM1 output pins, the /PWMn ration is "LOW=PWMn : HIGH=255-PWMn"
The PWMn values are compared to a 255-step counter (step-rate as defined by PWMP Prescaler, see below) and LOW is output if the counter is less or equal than PWMn. Writes to PWMn are immediately compared and /PWNn outputs are updated (without having to wait for the completion of the current counter period).

FEh - PWMP - Pulse Width Prescaler Frequency Control (Read/Write)
Specifies the step-rate (in relation to the system clock), which is shared for both PWM0 and PWM1. The step rate is:
```
  fSTEP = fCLK / 2 / (PWMP+1)
```
As each LOW:HIGH period consist off 255 steps, the repetition frequency is:
```
  fPWM  = fCLK / 510 / (PWMP+1)
```
Ie. the required PWMP value for specific frequency would be calculated as:
```
  PWMP  = fCLK / 510 / fPWM - 1
```
With 16MHz system clock, fPWM may range from 123Hz (FFh) to 31kHz (00h).

How to Turn off Pulse output
Setting PWMn to 00h or FFh will result in zero LOW- or HIGH-time, and in result, the /PWMn output will be constant HIGH (00h) or LOW (FFh).
By this method, /PWM0 and/or /PWM1 can be used as normal ON/OFF outputs.

Pulse Wave Example
Assuming the following settings:

```
  fCLK = 16MHz       ;System clock
  PWMP = 30          ;fPWM  = 16MHz/510/31 = 1012Hz
  PWM0 = 64          ;ratio = 64:191  = 40h:BFh
  PWM1 = 128         ;ratio = 128:127 = 80h:7Fh
```

Would result in:

```
  Repitition Rate:  <------><------><------>  each LOW:HIGH=998us (1012Hz)
  Output at /PWM0:  __------__------__------  each LOW=248us : HIGH=740us
  Output at /PWM1:  ____----____----____----  each LOW=496us : HIGH=492us
```

The PWM outputs may be used to drive DC motors (at variable speed), or as dual DAC outputs (when using a high frequency, smoothed-down pulses will effectively appear like analogue levels).

Interrupts

SFRs : IEN0, IEN1, IP0, IP1 (and IPLAFF)

IRQs

```
  TCON.1           IEN0.0    EX0     External Interrupt 0
  TCON.5           IEN0.1    ET0     Timer 0
  TCON.3+PLLCON.5  IEN0.2    EX1     External Interrupt 1, or Seconds Interrupt
  TCON.7           IEN0.3    ET1     Timer 1
  S0CON.0+1        IEN0.4    ES0     SIO0 (UART)
  S1CON.3          IEN0.5    ES1     SIO1 (I2C)
  ADCON.4          IEN0.6    EAD     ADC
  -                IEN0.7    EA      Global Enable (0=Disable all Interrupts)
  TM2IR.0-3        IEN1.0-3  ECT0-3  T2 Capture 0-3
  TM2IR.4-6        IEN1.4-6  ECM0-2  T2 Compare 0-2
  TM2IR.7,TM2CON.4 IEN1.7    ET2     T2 Overflow 16bit or Byte
```

Interrupt Vector Addresses

```
  Address  Prio  Name    Expl.
  0003h    1     X0      External Interrupt 0
  000Bh    4     T0      Timer 0 Overflow
  0013h    7     X1/SEC  External Interrupt 1 or Seconds Interrupt
  001Bh    10    T1      Timer 1 Overflow
  0023h    13    S0      SIO0 (UART/RS232) Send or Receive
  002Bh    2     S1      SIO1 (I2C)
  0033h    5     CT0     External Interrupt 2 with Timer 2 Capture 0
  003Bh    8     CT1     External Interrupt 3 with Timer 2 Capture 1
  0043h    11    CT2     External Interrupt 4 with Timer 2 Capture 2
  004Bh    14    CT3     External Interrupt 5 with Timer 2 Capture 3
  0053h    3     ADC     ADC Completion
  005Bh    6     CM0     Timer 2 Compare 0
  0063h    9     CM1     Timer 2 Compare 1
  006Bh    12    CM2     Timer 2 Compare 2
  0073h    15    T2      Timer 2 Overflow
```

Interrupt Priority 1=Highest, 15=Lowest.
Address 0000h is the Reset vector, which should have max. priority.

A8h - IEN0/IEC - Interrupt Enable Register 0 (Read/Write, Bit-addressable)

```
  Bit  Name    Expl. (0=Disable, 1=Enable)
  0    EX0     External Interrupt 0
  1    ET0     Timer 0
  2    EX1     External Interrupt 1, or Seconds Interrupt
  3    ET1     Timer 1
  4    ES0     SIO0 (UART)
  5    ES1     SIO1 (I2C)
  6    EAD     ADC
  7    EA      Global Enable (0=Disable all Interrupts)
```

E8h - IEN1 - Interrupt Enable Register 1 (Read/Write, Bit-addressable)

```
  Bit  Name    Expl. (0=Disable, 1=Enable)
  0-3  ECT0-3  T2 Capture 0-3
  4-6  ECM0-2  T2 Compare 0-2
  7    ET2     T2 Overflow
```

B8h - IP0/IPC - Interrupt Priority Register 0 (Read/Write, Bit-addressable)

```
  Bit  Name    Expl. (0=Low, 1=High)    Normal
  0    PX0     External Interrupt 0     1
  1    PT0     Timer 0                  4
  2    PX1     External Interrupt 1     7
               or Seconds Interrupt     7
  3    PT1     Timer 1                  10
  4    PS0     SIO0 (UART)              13
  5    PS1     SIO1 (I2C)               2
  6    PAD     ADC                      3
  7    -       Reserved for future use  -
```

F8h - IP1 - Interrupt Priority Register 1 (Read/Write, Bit-addressable)

```
  Bit  Name    Expl. (0=Low, 1=High)    Normal
  0-3  PCT0-3  T2 Capture 0-3           5,8,11,14
  4-6  PCM0-2  T2 Compare 0-2           6,9,12
  7    PT2     T2 Overflow              15
```

'IPLAFF' - Interrupt Priority Level Active Flipflops (Internal, non-SFR)

This is an internal 2bit register, it is not part of the SFR area. The P8xCE558 data sheet didn't described this register very well, the names IPLAFF, IPLAL, IPLAH, and most of the general description below are raw guesswork and might be (in-)correct (???)

```
  Bit  Name    Expl.
  0   'IPLAL'  Low Priority Interrupt Active  (1=Disables all low-prio IRQs)
  1   'IPLAH'  High Priority Interrupt Active (1=Disables all IRQs)
```

When an interrupt of low or high priority is executed, the respective flipflop (IPLAL or IPLAH) becomes set - this disables all other interrupts of same or lower priority. When the interrupt handler returns (by RETI instruction), the previous state is restored by clearing IPLAH (if it was set), or (otherwise) by clearing IPLAL.

IRQ Flags

The interrupt request flags are located in the separate Timer, Serial, A/D Converter, etc. control registers. Whereas external interrupt 0-1 are controlled by bits in Timer 0-1 control register, and external interrupt 2-5 by Timer 2 control registers. For Seconds interrupt see PLLCON in SYS Chapter.

SYS Chip Control Registers

SFRs : PCON, PLLCON, XRAMP, FMCON

87h - PCON - Power Control Register (Read/Write)

```
  Bit Name  Expl.
  0   IDL   Idle Mode                      (0=Normal, 1=Enter Idle Mode)
  1   PD    Power-Down (only if /EW=HIGH)  (0=Normal, 1=Enter Power Down Mode)
  2-3 GF0-1 General Purpose Flags          (Allowed to be used by software)
  4-6 -     Unused (except on special revisions, see below)
  7   SMOD  Double Baudrate Timer Divider in UART mode 1-3 (0=Div 32, 1=Div 16)
 P8xCE558:
  4   WLE   Watchdog Load Enable           (0=Lock, 1=Enable Write to Timer 3)
  5   RFI   Reduced Radio Frequency Interference (1=Suppress unused ALE pulses)
  6   ARD   AUX-RAM Disable        (0=Enable AUX-RAM, 1=Enable External memory)
 80C32/52 and up:
  4   POF   Power Off Flag (uh, is that a status bit, indicating coldboot?)
  5-6 -     Unused
```

For P8xCE558 in power reduction modes, the following is stopped (-), or kept operating (+), any interrupts caused by the active (+) components will terminate idle/powerdown mode.

```
  Mode        CPU PWM ADC T0  T1  T2  T3 UART I2C INT0 INT1 SECINT
  Idle Mode   -   -   -   +   +   -   +   +   +   +    +    +
  Power Down  -   -   -   -   -   -   N/A -   -   +    +    (RUN32)
```

When stopped (-), the CPU is halted, PWM is reset (output HIGH), ADC aborts current loop, timer 2 is stopped and reset (external interrupts 2-5 are disabled).

F9h - PLLCON - PLL Control Register (Read/Write)

All bits in this register working with XTAL3,4 Oscillator Circuit only, that is, when operating the chip by 32kHz crystal (SELXTAL1=GND) only.

```
  Bit  Name     Expl.
  0-4  FSEL.0-4 System Clock Frequency Selection (default=0Dh/11.01MHz)
  5    SECINT   Seconds Interrupt Flag   (Automatically set once per second)
  6    ENSECI   Seconds Interrupt Enable (1=Enable; INT1 must be enabled also)
  7    RUN32    32kHz oscillator during Power Down  (0=Halted, 1=Kept Running)
```

The SECINT flag can be cleared only by writing "0" to SECINT, also, the flag
may be set manually by writing "1".

Possible System Clock Selections (all capable of generating standard RS232 baudrates of 1200, 2400, 4800, 9600, 19200 Bauds with UART and Timer 1):

```
  0Bh=15.73MHz  0Dh=11.01MHz  0Fh=7.68MHz  11h=5.51MHz  13h=3.93MHz
  0Ch=12.58MHz  0Eh=9.44MHz   10h=6.29MHz  12h=4.72MHz  0h-0Ah,14h-1Fh=Reserved
```

Always recurse the following steps when changing the System Clock Frequency:

```
  From HIGH to LOW frequencies:
  First change FSEL.4-2, then FSEL.0-1 (and then better wait 1ms ?).
  From LOW to HIGH frequencies:
  First change FSEL.0-1, then wait 1ms, then change FSEL.2-4.
```

In detail, FSEL.0-1 are selecting the internal fCCO frequency of 32, 38, 44, or 50 MHz, changing FSEL.0-1 may lock the CPU for up to 10ms, and timing critical operations should not be executed within the following 1ms stabilization phase. FSEL.2-4 are dividing the above fCCO frequency into the actual system clock frequency, changing FSEL.2-4 recovers within 1us.

FAh - XRAMP - AUX-RAM Page Register (Read/Write)

Specifies address bits 8-9 for 8bit "MOVX @Ri" instructions for internal AUX-RAM (size 300h bytes). XRAMP is used only when ARD (AUX-RAM Disable, PCON.6) is cleared, otherwise, when ARD is set, a raw 8bit address is output to external memory.

```
  Bit  Name      Expl.
  0-1  XRAMP0-1  AUX-RAM Page Selection (0-2, 3=Reserved)
  2-7  -         Not used / Reserved    (always write "0" to these bits)
```

Note: The 16bit "MOVX @DPTR" instructions are not affected by XRAMP.

When ARD=0, DPTR=0..2FFh addresses AUX-RAM, and 300h..FFFFh addresses external memory. When ARD=1, all DPTR=0..FFFFh will access external memory.

FBh - FMCON - FEEPROM Control Register (Read/Write) - P89CE558 Only

This register does not directly control FEEPROM programming, instead, it is basically to be used as a 'key' which allows to read/write/erase FEEPROM memory by using the BIOS-functions in BOOT ROM, see FEEPROM chapter for details.

```
  Bit  Name     Expl.
  0-3  FCB0-3   Function Code
  4    -        Reserved, always write "0"
  5    HV       High Voltage Indication - Read Only
                (Set while high voltage for write/erase operation is present)
  6-7  UBS0-1   User/Boot Memory Selection
```

User/Boot Memory Selection

```
  0  User memory mapped from 0 to 64K
  1  User memory mapped from 0 to 63K, Boot ROM from 63K to 64K
  2  Reserved/Internal
  3  Reserved/Internal
  (User memory may be internal and/or external memory)
```

Function Codes

```
  00h  Value after Reset
  05h  Byte Write or Byte Read/Verify
  0Ch  Page Erase  (32 bytes boundaries)
  03h  Block Erase (256 bytes boundaries)
  0Ah  Full Erase  (32 Kbytes)
```

CPU Microprocessor

SFRs : A/ACC, B, DPTR (DPH:DPL), SP, PSW -- RAM : R0-R7 (BANK 0-3) -- PC

CPU Registers and Flags

SFRs : A/ACC, B, DPTR (DPH:DPL), SP, PSW -- RAM : R0-R7 (BANK 0-3) -- PC

E0h - A/ACC - Accumulator (Read/Write, Bit-Addressable)
General purpose register, used as accumulator for various maths instructions.

F0h - B - The B Register (Read/Write, Bit-Addressable)
Even though "B" is a pretty nice short name, use of this register will not actually result in 'short' code - the register is NOT used as implied operand by any

opcodes (except MUL/DIV), thus using R0-R7 instead of B will often result in smaller (and sometimes faster) program code. However, possible advantages are that the register is bit-addressable, and that it is not disturbed by register-bank switching.

00h-07h - R0-R7 - Registers R0-R7 Bank 0 (Read/Write) (default)
08h-0Fh - R0-R7 - Registers R0-R7 Bank 1 (Read/Write)
10h-17h - R0-R7 - Registers R0-R7 Bank 2 (Read/Write)
18h-1Fh - R0-R7 - Registers R0-R7 Bank 3 (Read/Write)
R0-R7 are eight general purpose registers, R0-R1 can be also used for addressing RAM and XRAM. The registers are usually mapped to the first bytes of RAM, so those RAM addresses are somewhat reserved for those registers.
A rather uncommon feature is allowing to map R0-R7 to four different "banks" (via bits in PSW register). The confusing part is that the bank switching will affect only opcodes with implied R0-R7 operands; for example, there is no "PUSH R0" opcode, but one could "PUSH [00h]" instead, however, that method would always push R0.bank0, regardless of the currently selected bank).

82h/83h - DPL/DPH - DPTR - Data Pointer (Read/Write)
Intended for 16bit memory addressing, DPTR can be used as implied operand by a handful of opcodes, additionally, lower and upper 8bits can be separately accessed by DPL and DPH direct operands.

81h - SP - Stack Pointer (Read/Write)
8bit stack pointer into internal RAM - chips with 256 bytes internal RAM (eg. P8xCE558) may use the whole 8bit range, chips with 128 bytes (eg. 8051) may use only 7bit range.
The stack is INCREMENTED when writing data to it (unlike as in most other CPUs). PUSH/POP store/load 8bit data, CALL/RET (and INT/RETI) store/load 16bit data (LSB at smaller address). Before writing data, SP is incremented once, ie. initialize as "MOV SP,#stackbase-1".
Keep in mind that RAM 00h-1Fh is used for registers R0-R7 in banks 0-3, the default/reset value (SP=07h, eg. 08h and up) conflicts with register banks 1-3.

N/A - PC - Program Counter
16bit Instruction pointer, incremted each time when reading an opcode or parameter byte from program code. Any addressing relative to PC (eg. SJMP, ACALL, CJNE, JZ, @A+PC, etc.) are originated from the end of the current opcode (=beginning of next opcode), the pushed return address from CALLs points to the next opcode as well.
Unlike all other registers, PC is NOT stored in the SFR-area.

D0h - PSW - Program Status Word (Flags) (Read/Write, Bit-Addressable)
```
  Bit  Name   Expl.
  0    P      Parity of Accumulator (0=Even, 1=Odd) - Read Only
  1    F1     Reserved in 8031/51 models
  2    OV     Overflow Flag         (0=No Overflow, 1=Overflow)
  3-4  RS0-1  Register Bank Select  (Bank 0-3, for registers R0-R7)
  5    F0     User Flag 0           (May be used for whatever purpose)
  6    AC     Auxiliary Carry Flag  (For "DA A" opcode)
  7    CY     Carry Flag            (0=No Carry, 1=Carry)
```

Instructions that affect flag settings

```
Instruction CY OV AC    Instruction CY OV AC    Instruction CY OV AC
ADD         X  X  X     CJNE        X  -  -     ANL  C,bit  X  -  -
ADDC        X  X  X     RRC         X  -  -     ANL  C,/bit X  -  -
SUBB        X  X  X     RLC         X  -  -     ORL  C,bit  X  -  -
MUL         0  X  -     SETB C      1  -  -     ORL  C,/bit X  -  -
DIV         0  X  -     CLR  C      0  -  -     MOV  C,bit  X  -  -
DA          X  -  -     CPL  C      X  -  -
```

CPU Arithmetic Operations

```
Mnemonic        Nocash     Bytes/Cycles    Description
ADD  A,Rn       ADD  A,Rn      1/1  Add register to A
ADD  A,direct   ADD  A,[nn]    2/1  Add direct byte to A
ADD  A,@Ri      ADD  A,[Ri]    1/1  Add indirect RAM to A
ADD  A,#data    ADD  A,nn      2/1  Add immediate to A
ADDC A,Rn       ADC  A,Rn      1/1  Add register to A with Carry
ADDC A,direct   ADC  A,[nn]    2/1  Add direct byte to A with Carry
ADDC A,@Ri      ADC  A,[Ri]    1/1  Add indirect RAM to A with Carry
ADDC A,#data    ADC  A,nn      2/1  Add immediate to A with Carry
SUBB A,Rn       SBC  A,Rn      1/1  Subtract register from A with borrow
SUBB A,direct   SBC  A,[nn]    2/1  Subtract direct byte from A with borrow
SUBB A,@Ri      SBC  A,[Ri]    1/1  Subtract indirect RAM from A with borrow
SUBB A,#data    SBC  A,nn      2/1  Subtract immediate from A with borrow
INC  A          INC  A         1/1  Increment A
INC  Rn         INC  Rn        1/1  Increment register
INC  direct     INC  [nn]      2/1  Increment direct byte
INC  @Ri        INC  [Ri]      1/1  Increment indirect RAM
INC  DPTR       INC  DPTR      1/1  Increment data pointer (16 bit value)
DEC  A          DEC  A         1/1  Decrement A
DEC  Rn         DEC  Rn        2/1  Decrement register
DEC  direct     DEC  [nn]      1/1  Decrement direct byte
DEC  @Ri        DEC  [Ri]      1/2  Decrement indirect RAM
MUL  AB         MUL  A,B       1/4  Multiply A & B (BA = A * B)
DIV  AB         DIV  A,B       1/4  Divide A by B (A=A/B, B=A MOD B)
DA   A          DAA  A         1/1  Decimal Adjust (after 'BCD+BCD'operation)
```

CPU Logical Operations

```
Mnemonic           Nocash     Bytes/Cycles    Description
ANL  A,Rn          AND  A,Rn      1/1  AND register to A
ANL  A,direct      AND  A,[nn]    2/1  AND direct byte to A
```

ANL A,@Ri	AND A,[Ri]	1/1	AND indirect RAM to A
ANL A,#data	AND A,nn	2/1	AND immediate to A
ANL direct,A	AND [nn],A	2/1	AND A to direct byte
ANL direct,#data	AND [nn],nn	3/2	AND immediate to direct byte
ORL A,Rn	OR A,Rn	1/1	OR register to A
ORL A,direct	OR A,[nn]	2/1	OR direct byte to A
ORL A,@Ri	OR A,[Ri]	1/1	OR indirect RAM to A
ORL A,#data	OR A,nn	2/1	OR immediate to A
ORL direct,A	OR [nn],A	2/1	OR A to direct byte
ORL direct,#data	OR [nn],nn	3/2	OR immediate to direct byte
XRL A,Rn	XOR A,Rn	1/1	Exclusive-OR register to A
XRL A,direct	XOR A,[nn]	2/1	Exclusive-OR direct byte to A
XRL A,@Ri	XOR A,[Ri]	1/1	Exclusive-OR indirect RAM to A
XRL A,#data	XOR A,nn	2/1	Exclusive-OR immediate to A
XRL direct,A	XOR [nn],A	2/1	Exclusive-OR A to direct byte
XRL direct,#data	XOR [nn],nn	3/2	Exclusive-OR immediate to direct byte
CLR A	CLR A	1/1	Clear A
CPL A	CPL A	1/1	Complement A
RL A	ROL A	1/1	Rotate A Left
RLC A	RCL A	1/1	Rotate A Left through the Carry
RR A	ROR A	1/1	Rotate A Right
RRC A	RCR A	1/1	Rotate A Right through the Carry
SWAP A	SWAP A	1/1	Swap nibbles within A (4bit rotate)
CLR C	CLR C	1/1	Clear carry
CLR bit	CLR bit	2/1	Clear direct bit
SETB C	SET C	1/1	Set carry
SETB bit	SET bit	2/1	Set direct bit
CPL C	CPL C	1/1	Complement carry
CPL bit	CPL bit	2/1	Complement direct bit
ANL C,bit	AND C,bit	2/2	AND direct bit to Carry
ANL C,/bit	AND C,/bit	2/2	AND complement of direct bit to Carry
ORL C,bit	OR C,bit	2/2	OR direct bit to Carry
ORL C,/bit	OR C,/bit	2/2	OR complement of direct bit to Carry

CPU Data Transfer

Mnemonic	**Nocash**	**Bytes/Cycles**	**Description**
MOV A,Rn	MOV A,Rn	1/1	Move register to A
MOV A,direct	MOV A,[nn]	2/1	Move direct byte to A
MOV A,@Ri	MOV A,[Ri]	1/1	Move indirect RAM to A
MOV A,#data	MOV A,nn	2/1	Move immediate to A
MOV Rn,A	MOV Rn,A	1/1	Move A to register
MOV Rn,direct	MOV Rn,[nn]	2/2	Move direct byte to register
MOV Rn,#data	MOV Rn,nn	2/1	Move immediate to register
MOV direct,A	MOV [nn],A	2/1	Move A to direct byte

MOV direct,Rn	MOV [nn],Rn	2/2	Move register to A
MOV direct,direct	MOV [nn],[nn]	3/2	Move direct byte to direct byte
MOV direct,@Ri	MOV [nn],[Ri]	2/2	Move indirect RAM to direct byte
MOV direct,#data	MOV [nn],nn	3/2	Move immediate to direct byte
MOV @Ri,A	MOV [Ri],A	1/1	Move A to indirect RAM
MOV @Ri,direct	MOV [Ri],[nn]	2/2	Move direct byte to indirect RAM
MOV @Ri,#data	MOV [Ri],nn	2/1	Move immediate to indirect RAM
MOV DPTR,#data16	MOV DPTR,nnnn	3/2	Load data pointer with 16bit constant
MOVC A,@A+DPTR	MOVC A,[A+DPTR]	1/2	Move code byte relative to DPTR to A
MOVC A,@A+PC	MOVC A,[A+PC]	1/2	Move code byte relative to PC to A
MOVX A,@Ri	MOVX A,[Ri]	1/2	Move external RAM (8bit addr) to A
MOVX A,@DPTR	MOVX A,[DPTR]	1/2	Move external RAM (16bit addr) to A
MOVX @Ri,A	MOVX [Ri],A	1/2	Move A to external RAM (8bit addr)
MOVX @DPTR,A	MOVX [DPTR],A	1/2	Move A to external RAM (16bit addr)
PUSH direct	PUSH [nn]	2/2	Increment SP, push direct byte to SP
POP direct	POP [nn]	2/2	Pop direct byte from SP, decrement SP
XCH A,Rn	XCHG A,Rn	1/1	Exchange register with A
XCH A,direct	XCHG A,[nn]	2/1	Exchange direct byte with A
XCH A,@Ri	XCHG A,[Ri]	1/1	Exchange indirect RAM with A
XCHD A,@Ri	XCHD A,[Ri]	1/1	Exchange lower digit (4bit) with A
MOV C,bit	MOV C,bit	2/1	Move direct bit to Carry
MOV bit,C	MOV bit,C	2/2	Move Carry to direct bit

CPU Program Branching

Mnemonic	**Nocash**	**Bytes/Cycles**	**Description**
ACALL addr11	ACALL addr11	2/2	Absolute subroutine call
LCALL addr16	LCALL addr16	3/2	Long subroutine call
RET	RET	1/2	Return for subroutine
RETI	RETI	1/2	Return for interrupt
SJMP rel	SJMP rel	2/2	Short jump (8bit relative)
AJMP addr11	AJMP addr11	2/2	Absolute jump (11bit absolute)
LJMP addr16	LJMP addr16	3/2	Long jump (16bit long)
JMP @A+DPTR	JMP A+DPTR	1/2	Jump indirect relative to DPTR
JZ rel	JZ A,rel	2/2	Jump if A is zero
JNZ rel	JNZ A,rel	2/2	Jump if A is not zero
CJNE A,direct,rel	JNE A,[nn],rel	3/2	Cmp direct byte to A, jump if not eq
CJNE A,#data,rel	JNE A,nn,rel	3/2	Cmp imm to A, jump if not eq
CJNE Rn,#data,rel	JNE Rn,nn,rel	3/2	Cmp imm to register, jump if not eq
CJNE @Ri,#data,rel	JNE [Ri],nn,rel	3/2	Cmp imm to indirect, jump if not eq
DJNZ Rn,rel	DJNZ Rn,rel	3/2	Decrement register, jump if not zero
DJNZ direct,rel	DJNZ [nn],rel	3/2	Decrement direct, jump if not zero
JC rel	JC rel	2/2	Jump if Carry is set
JNC rel	JNC rel	2/2	Jump if Carry is not set
JB bit,rel	JNZ bit,rel	3/2	Jump if direct bit is set

```
  JNB  bit,rel       JZ   bit,rel     3/2  Jump if direct bit is not set
  JBC  bit,rel       JNZ0 bit,rel     3/2  Jump if direct bit is set, clear bit
  (INT vector)       (INT vector)     -/2? Interrupt (LCALL to vector address)
```

The assembler automatically converts JMP and CALL into best matching opcodes. LJMP/LCALL is choosen in case of forward references. AJMP/ACALL works only inside of the current 800h-page. SJMP covers a range of -128..+127 bytes, which may cross page boundaries.

CPU Notes

Notes on instruction set and addressing modes

```
  Rn        - Register R7-R0 of the currently selected register bank
              (The register bank is selected by PSW.3 and PSW.4)
              (Note that various opcodes support only <direct> operands,
              but not <Rn> operands (eg. PUSH/POP). In such cases, the
              assembler converts R0-R7 into direct addresses 00h-07h,
              which is forcefully accessing register bank 0 only)
  direct    - 8-bit internal data locations'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 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 in 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 -128 to +127
              bytes relative to first byte of following the instruction
  bit       - Direct addressed bit in internal data RAM or special
              function register (SFR)
```

Hardware information (Instruction set)

One cycle equals to 12 oscillator periods.

Flash EEPROM

Flash EEPROM is included in P89CE558 chips only.

FEEPROM User Access

FFBAh - BYTE_READ
FFADh - BYTE_WRITE
FFAAh - PAGE_ERASE
FFA5h - BLOCK_ERASE
FFA0h - FULL_ERASE
FC07h - SERIAL_BOOT

All functions may be invoked from inside of internal or external memory. Interrupts should be disabled before the call, and FMCON should be reset to zero immediately after the call (that is, as far as I understand, both meant to be for security against piracy). Aside from below return values, all registers remain unchanged.

FFBAh - BYTE_READ - Read one byte from FEEPROM
```
  In:  FMCON=45h, DPTR=Byte Address
  Out: FMCON=15h, A=DATA, DPTR=Unchanged
```
Execution time should be only a couple of clock cycles, however, when internal memory is enabled (/EA=1), then the same result can be gained more easily and faster by a simple "MOVC A,@DPTR" instruction.

FFADh - BYTE_WRITE - Write one byte to FEEPROM
```
  In:  FMCON=45h, A=DATA, DPTR=Byte Address
  Out: FMCON=15h, A=DATA (read-back), DPTR=Unchanged
```
Execution time is 2.5ms, the destination must have been previously erased, the returned DATA contains read-back data from FEEPROM, a write failure may be detected by comparing the original and returned DATA.

FFAAh - PAGE_ERASE - Erase 32 Bytes of FEEPROM
```
  In:  FMCON=4Ch, DPTR=Page Address, lower 5bits ignored
  Out: FMCON=1Ch, A=08h, DPTR=Unchanged, except that lower 5bits reset
```
Execution time is 5ms, the erased memory will be FFh-filled.

FFA5h - BLOCK_ERASE - Erase 256 Bytes of FEEPROM
```
  In:  FMCON=43h, DPTR=Block Address, lower 8bits (DPL) ignored
  Out: FMCON=13h, ACC=02h, DPTR=Unchanged, except that lower 8bits reset
```

Execution time is 5ms, the erased memory will be FFh-filled.

FFA0h - FULL_ERASE - Erase whole 32 KBytes of FEEPROM

```
  In:  FMCON=4Ah
  Out: FMCON=1Ah, ACC=0Ah, DPTR=0018h
```

Execution time is 5ms, the erased memory will be FFh-filled.
For obvious reason, a full erase should be attempted only if the return address is in external memory.

FC07h - SERIAL_BOOT - Download Code/Data via UART/RS232 into FEEPROM

```
  In:  FMCON=40h, Interrupt Registers, Stack Pointer, Timer 0, UART, P3.0-1
  Out: Does never return - system must be manually reset after transfer.
```

Execution time is 2.5ms per programmed byte (or slower when using less than 9600 Bauds). Because the function does not return, it should be invoked by JMP rather than CALL. Aside from FMCON, all of the above listed registers must be in reset state. Normally, this function is invoked by external signals, see chaper "FEEPROM Serial Programming" for details.

For (unimportant) details about the FMCON registers, see SYS chapter.

FEEPROM Security

FEEPROM Parallel Programming

It is possible to program/erase the FEEPROM in the P89CE558 by parallel programming, similar as a normal EPROM, high programming voltages aren't required. The programming time is 2.5ms per byte (400 bytes/second), programming the whole 32KBytes would take approximately 82 seconds. However, parallel programming requires rather extensive connections:

```
  15 address lines,
  8 data lines,
  9 control lines,
  and one 4-6MHz oscillator at XTAL1/XTAL2
```

For details refer to data sheet. In most cases it'd be less complicated to chose serial programming (see next chapter), and, when using a serial transfer rate of at least 9600 Bauds, programming time should be quite as fast as parallel programming.

FEEPROM Serial Programming

Transfer Record Format

Binary data must be converted into Intel Hex Object Format (separate records of ASCII strings which are formatted as ":BCAAAATTHH..HHCC").

```
  :     Record Start character
  BC    Byte Count (number of HH data bytes in this record, 00..FF)
  AAAA  Destination address of first byte of this record (0000..7FFF)
  TT    Record Type (00=data record, 01=end record)
  HH    Data Byte(s)
  CC    Record Checksum (CC = 00-BC-AA-AA-TT-HH-HH-HH-...-HH)
```

Any data between ending "CC" and next following ":" will be ignored (ie. optional ending carriage returns/linefeeds will not cause transmission errors).

Sending Records

Send any number of data records, each record is allowed to contain any number of bytes (BC=00..FF), destination addresses (AAAA) are not required to be sent in sequential order, there is no need to care about page boundaries in the FEEPROM chip (a record may cross page boundaries, and, when changing only some byte(s) inside if a page, the unchanged bytes are internally saved in RAM before erasing that page, both changed and unchanged bytes are then (re-) written to FEEPROM).
Send the end record once when all data records have been sent, the end record must be always ":00000001FF", ie. BC=00, AAAA=0000, TT=01, with proper CC checksum as usually.

Serial Communication

Data must be transferred as 1 startbit, 8 databits, and at least 1 stopbit.
The following character messages are sent from P8xCE559 to master.

```
  "."  Acknowledges record type TT=00 received.
  "X"  Error - Bad CC checksum.
  "Y"  Error - Bad TT record type.
  "Z"  Error - Buffer overflow (Check Xon/Xoff)
  "R"  Error - Verification Error (of last byte written).
  "V"  End record (TT=01) received, and FEEPROM programming completed.
  Xoff Busy. Master may not send further data.           ;Xoff=chr(13h)
  Xon  End of Busy period. Master may continue sending.  ;Xon =chr(???)
```

No messages are sent if the baud rate for the first ":" character couldn't be detected, valid baudrates are (provided that the system clock is within specified min/max ranges):

```
  Baudrate  1200  2400  4800  9600  19200  Bauds
  fCLKmin   1     2     4     7.9   15.7   MHz
  fCLKmax   3.6   7.3   14.7  29.5  59     MHz
```

Notes: When using 'variable' system clock (generated from 32.768kHz oscillator at XTAL3-4, SELXTAL1=0), then system clock is initialized at 11.01MHz (thus only 4800 or 9600 bauds will be recognized).
Otherwise, when using fixed system clock (generated from oscillator at XTAL1-2, SELXTAL1=1), only 3.5MHz..16MHz are actually supported by the hardware.
A baudrate of 19200 bauds would not actually increase the performance, the internal programming time is 2.5ms/byte (400 bytes/sec), and 9600 bauds (approx. 960 chars/sec = approx. 450 bytes/sec) would be thus more than fast enough.

CIR Basic Connection Circuits

CIR Reset Circuit

RSTIN-Pin (Reset Input)
Used for Power-on reset (and/or external reset "button", etc).
Reset is active when the input is HIGH (!)

RSTOUT-Pin (Reset Output)
Upon sensing an incoming reset at RSTIN (or when generating an internal reset caused by timer 3 watchdog overflow), a reset (HIGH) signal is output to RSTOUT - this should be used to reset any peripherals which are connected to the CPU.

8xCE558 Power-On Reset Circuit

```
  +5V ----[]|---- RSTIN
           +   -
```

Use at least 2.2uF capacitor when using HF-oscillator at XTAL1/2,
use 0.1uF capacitor when using PLL-osciallator at XTAL3/4.

After reset, program execution starts at 0000h, and most SFR registers are reset. Internal RAM is not initialized - its contents are undefined upon power-on, and are kept unchanged upon 'warm' reset.

CIR Oscillator (System Clock)

```
  XTAL3,4 Oscillator Circuit,   XTAL1,2 Oscillator Circuit    XTAL1 External
  PLL Oscillator+Seconds Timer  (Standard 80C51 compatible)   Clock Input
  SELXTAL1    XTAL3 XTAL4       XTAL1       XTAL2   SELXTAL1  XTAL1 SELXTAL1
     |          |     |           |           |       |         |      |
    GND         +-|O|-+           +-||-GND-||-+      +5V        |     +5V
  Software-selectable system      |           |                 EXT.CLK,
  clock rates of 3.93-15.73MHz    +----|O|----+   3.5-16MHz     3.5-16MHz
  Use external 32768Hz crystal  Use 22pF capacitors, quartz   Leave XTAL2
  without external capacitors.  crystal or ceramic resonator. not connected.
```

Note:
One instruction cycle equals to 12 oscillator periods, that is, assuming a system clock of approximately 12MHz, a the execution time for one NOP opcode will be 1us.

CIR Pin-Outs

Pinning diagram for the 8051 family

```
                  __________ __________
                 |          \/          |
          P1.0   | 01                40 |    VCC
          P1.1   | 02                39 |    P0.0   AD0
          P1.2   | 03                38 |    P0.1   AD1
          P1.3   | 04                37 |    P0.2   AD2
          P1.4   | 05                36 |    P0.3   AD3
          P1.5   | 06                35 |    P0.4   AD4
          P1.6   | 07                34 |    P0.5   AD5
          P1.7   | 08                33 |    P0.6   AD6
      RST,VPD*   | 09      8051      32 |    P0.7   AD7
    RXD   P3.0   | 10                31 |    /EA
    TXD   P3.1   | 11                30 |    ALE
  /INT0   P3.2   | 12                29 |    /PSEN
  /INT1   P3.3   | 13                28 |    P2.7   AD15
     T0   P3.4   | 14                27 |    P2.6   AD14
     T1   P3.5   | 15                26 |    P2.5   AD13
    /WR   P3.6   | 16                25 |    P2.4   AD12
    /RD   P3.7   | 17                24 |    P2.3   AD11
         XTAL2   | 18                23 |    P2.2   AD10
         XTAL1   | 19                22 |    P2.1   AD9
           VSS   | 20                21 |    P2.0   AD8
                 |______________________|
```

(*) VPD applicable to NMOS versions only

Pinning for P8xCE558
80 Pins SMD - Plastic Quat Flat Pack, 80 leads QFP80 (SOT318)

```
  1   AVref-       21  P4.2,CMSR2      41  P3.0,RXD    61  P2.6,A14
  2   AVref+       22  P4.3,CMSR3      42  P3.1,TXD    62  P2.7,A15
  3   AVss1        23  RSTOUT          43  P3.2,/INT0  63  /PSEN
  4   AVdd1        24  P4.4,CSMR4      44  P3.3,/INT1  64  ALE,/WE (*)
  5   P5.7,ADC7    25  P4.5,CSMR5      45  P3.4,T0     65  /EA
  6   P5.6,ADC6    26  P4.6,CMT0       46  P3.5,T1     66  Vdd4
  7   P5.5,ADC5    27  P4.7,CMT1       47  P3.6,/WR    67  Vss4
  8   P5.4,ADC4    28  Vdd2            48  P3.7,/RD    68  P0.7,AD7
  9   P5.3,ADC3    29  Vss2            49  n.c.        69  P0.6,AD6
  10  P5.2,ADC2    30  RSTIN           50  n.c.        70  P0.5,AD5
  11  P5.1,ADC1    31  P1.0,CT0I,INT2  51  XTAL2       71  P0.4,AD4
  12  P5.0,ADC0    32  P1.1,CT1I,INT3  52  XTAL1       72  P0.3,AD3
  13  Vss1         33  P1.2,CT2I,INT4  53  Vdd3        73  P0.2,AD2
  14  Vdd1         34  P1.3,CT3I,INT5  54  Vss3        74  P0.1,AD1
  15  ADEXS        35  P1.4,T2         55  P2.0,A8     75  P0.0,AD0
  16  /PWM0        36  P1.5,RT2        56  P2.1,A9     76  AVdd2
  17  /PWM1        37  P1.6            57  P2.2,A10    77  AVss2
  18  EW           38  P1.7            58  P2.3,A11    78  XTAL3
  19  P4.0,CMSR0   39  SCL             59  P2.4,A12    79  XTAL4
  20  P4.1,CMSR1   40  SDA             60  P2.5,A13    80  SELXTAL1
```
(*) "/WE" for P89CE558 only. "n.c."=not connected.

AUX External Hardware

Currently emulated hardware
No$x51 currently emulates a numeric keypad, and a 16x2 character LCD display:
AUX Numeric Keypad
AUX LCD Dot Matrix Module

Emulating other hardware
There are certainly millions of external devices which could be connected and combined in various ways, and emulating all of that would be impossible.
However, upon request (if somebody would pay for it) I'd be considering to emulate whatever required standard or non-standard hardware, including other displays, LEDs, keyboards, buttons, memory, elevators, machines, sensors, etc.

Remote controlled chip
The most obvious (and easiest to implement) way to debug external hardware would be to link the debugger to a real 8051/family chip by simple RS232 connection - assuming that your project already includes a RS232 interface, you'd not need any additional hardware (except for a small BIOS loaded into your (FE-)EPROM.
All program opcodes and internal timers would be kept emulated on the PC, but read/write accesses to digital I/O ports P0..P5 or to ADC sensor inputs would be transferred to/from real hardware.
This method would not be suitable for timing critical operations, and it'd become kinda slow when transferring more than 1000 bytes/second. Aside from that, it should allow to access external keyboards, displays, sensors, and other inputs and outputs...
If anybody is interested, please let me know.

Software plug-ins
Another solution would be to provide a plug-in interface which'd allow people to emulate their own hardware. I've not yet dealt with plug-ins though, and would definetly need some tips/examples on how to implement such a thing.

Customizing your own software

The emulator may be detected by software (if enabled in setup) by examing the content of the DPTR register directly after reset (value 0CA5h indicates no$x51 emulator).
For example, if your program requires a 4-digit LED display and two push buttons, then (when detecting the emulator) you may redirect input and output to the emulated 16x2 LCD display and numeric keypad.
Also, in case that your program 'hangs' in lack of incoming data from external inputs, then you may want to skip over these inputs when having detected the emulator.

AUX Numeric Keypad

Emulated Keyboard Interface
Currently emulates only access through <direct> operands.

```
  P4.7-5 Out  Select Row, 0=Select
  P4.4-1 Out  Always output "1" to these bits
  P4.4-1 In   Read currently selected keyboard row(s), 0=Pressed
  P4.0   -    Not used
```

Keys are mapped to following PC keys:

```
  0-9  -->  Keypad numbers and normal numbers
  *    -->  Keypad Enter and normal Enter
  #    -->  Keypad "." and Backspace
```

Keyboard Matrix

```
  Ports  P4.1  P4.2  P4.3  P4.4
  P4.7   "1"   "4"   "7"   "*"
  P4.6   "2"   "5"   "8"   "0"
  P4.5   "3"   "6"   "9"   "#"
```

AUX LCD Dot Matrix Module

Emulated LTN I/O Interface

```
  Instruction Output (RS=0): MOV DPH,#80h / MOVX @DPTR,A
  Data Output (RS=1, R/W=0): MOV DPH,#82h / MOVX @DPTR,A
  Data Input  (RS=1, R/W=1): MOV DPH,#83h / MOVX A,@DPTR
```

Currently emulates LTN 211R-10 (16 x 2 characters) only.
Special functions such like scrolling aren't yet understood.

LTN Instruction Set

```
  RS__R/W__D7__D6__D5__D4__D3__D2__D1__D0__Instruction___________
  0   0    0   0   0   0   0   0   0   1   Display Clear
  0   0    0   0   0   0   0   0   1   *   Cursor Home
  0   0    0   0   0   0   0   1  I/D  S   Entry Mode Set
  0   0    0   0   0   0   1   D   C   B   Display on/off Control
  0   0    0   0   0   1  S/C R/L  *   *   Cursor Display Shift
  0   0    0   0   1   DL  1   0   *   *   Function Set
  0   0    0   1  <-------- Acg -------->  CG RAM Address Set (0-3Fh)
  0   0    1  <------------ Add -------->  DD RAM Address Set (0-7Fh)
  0   1    BF <------------ Ac  -------->  Busy Flag/Address Read
  1   0    <-------- write data -------->  CG/DD RAM data write
  1   1    <-------- read  data -------->  CG/DD RAM data read
```

Whereas:

```
  I/D  0=Decrement,      1=Increment
  S    0=Display freeze, 1=Display shift
  D    0=Display off,    1=Display on
  C    0=Cursor off,     1=Cursor on
  B    0=Blinking off,   1=Character at Cursor position blinking
  S/C  0=Cursor move,    1=Display shift
  R/L  0=left shift,     1=right shift
  DL   0=4bits,          1=8bits
  BF   0=Not busy,       1=Internal operation busy
  *    Don't care (?)
```

DD-RAM Memory Map

LTN 111R-10 DD-RAM (16 x 1 characters):

```
  Line 1  00h 01h 02h 03h 04h 05h 06h 07h 40h ... 47h  (!)
```

LTN 211R-10 DD-RAM (16 x 2 characters):

```
  Line 1  00h 01h 02h 03h 04h 05h 06h 07h 08h ... 0Fh
  Line 2  40h 41h 42h 43h 44h 45h 46h 47h 48h ... 4Fh
```

LTN 242R-10 DD-RAM (40 x 2 characters):

```
  Line 1  00h 01h 02h 03h 04h 05h 06h 07h 08h ... 26h 27h
  Line 2  40h 41h 42h 43h 44h 45h 46h 47h 48h ... 66h 67h
```

CG-RAM Memory Map

Character Generator RAM for eight user-defined characters of 8 bytes each. Format of bitmap for each character not specified/unknown ???

Character Set (5x7 dots, excluding spacing between characters)

```
  00h-07h  CG RAM (1-8)
  08h-0Fh  CG RAM (1-8)
  10h-1Fh  Undefined
  20h-7Fh  Normal ASCII charcters (*)
  80h-9Fh  Undefined
  A0h-FFh  Japanese/European characters
```

(*) Except non-ascii 5Ch (yen instead "\"), 7Eh (arrow right instead "~"), and 7Fh (arrow left). Some of the E0h-FFh characters are sized 8 dots vertically and cannot be displayed properly on above listed display types.

AMT630A - Memory Map

CODE (opcode-fetches and MOVC reads)

```
  0000h..7FFFh  Fixed    (always 1st 32Kbytes of SPI FLASH memory)
  8000h..FFFFh  Mappable (usually 2nd 32Kbytes of SPI FLASH memory)
```

CPU clock is 6.75MHz (27MHz/4). The code is loaded from serial SPI bus to some cache. The CPU will be paused for around 43us upon cache misses, causing the overall CPU clock to drop to around 2.03MHz (as so when executing thousands of 1-cycle NOP opcodes).

RAM/SFR (data/registers accessed via MOV/ALU opcodes)

```
  00h..07h      Standard CPU registers R0-R7 (bank 0)
  08h..7Fh      Standard RAM
  80h..FFh      Extended RAM, 80C52-style (via indirect [R0],[R1],[SP])
  80h..FFh      Standard+Extended SFR Registers (via direct [imm])
```

XRAM (accessed via MOVX opcodes)

```
  0000h..07FFh  Extra RAM (2Kbytes)
  0800h..1FFFh  Mirror of SPI FLASH addresses 000800h..001FFFh (read only)
  2000h..2FFFh  Unknown (hardware status regs ?)     (read only)
  3000h..FAFFh  Unused (open-bus)
  FB00h..FBFFh  I/O Ports (OSD on-screen display)
  FC00h..FCFFh  I/O Ports (LCD screen ratio)
  FD00h..FDFFh  I/O Ports (Misc, ADC, PWM, PLL, PIN, FLASH, etc.)
  FE00h..FEFFh  I/O Ports (AV video input)
  FF00h..FFFFh  I/O Ports (LCD colors/brightness and IR Infrared)
```

OSD Memory (VRAM, write-only(?), accessed via FCxxh I/O ports, or FONT-DMA)

```
  BGMAP RAM: 200h entries (each 10bit character number plus 7bit attribute)
  FONT RAM:  1000h words (aka 2000h bytes, aka 8Kbytes)
  FONT ROM:  418 characters (16x22pix, 1bpp; uppercase text & exotic symbols)
```

SPI FLASH (accessed via FDxxh ports, and directly mapped to CPU memory space)

AMT630A boards are commonly fitted with 256-512Kbyte FLASH (though firmware uses less than 48Kbytes, plus two 1000h-byte sectors at E000h/F000h for saving user settings).
First 2x32Kbyte FLASH are usually mapped to CPU's CODE memory (16bit address space).
The whole FLASH can be accessed via FDxxh I/O ports (24bit address space).
The FDxxh I/O ports also have some FLASH DMA support.
Some FLASH snippet is also mirrored to XRAM memory space (unknown if one can somehow select WHICH snippet).

Further memory

The AMT630A chip has some memory buffer for resampling scanlines to actual TFT resolution (with the improper PAL60 support, one can actually see that the

hardware does memorize older scanlines).
Aside from firmware FLASH, there might be some Boot ROM (for initializing FLASH acccess, possible with 2bit/4bit databus, depending on the installed FLASH chip; 25Dxx vs 25Qxx).
SPI FLASH is somehow cached for reducing SPI bus traffic, details on cache size are unknown.

Exception Vectors in CODE memory

```
  0000h reset      firmware reset vector (and apparently watchdog, too)
  0003h infrared   firmware has IR infrared handler (in some versions) (ext.0)
  000Bh timer 0    firmware has dummy timer 0 reload handler
  0013h spi flash  firmware has no handler for this                    (ext.1)
  001Bh timer 1    firmware has timer 1 handler (sensing AV signal etc)
  0023h uart       firmware has no handler for this
  002Bh timer 2    firmware has no handler for this (80C52-style extension)
  0033h ?          firmware has no handler for this
  003Bh ? (if any) firmware has no handler for this (are there IEC/IPC bits?)
  0043h ADC        firmware acknowledges SFR_IO_xxx91h.bit4 and IO_ADC_status
  004Bh ?          firmware acknowledges SFR_IO_xxx91h.bit5
  0053h ?          firmware acknowledges SFR_IO_xxx91h.bit6
  005Bh framerate  firmware acknowledges SFR_IO_xxx91h.bit7 (vblank/vsync?)
  0063h timer 3+4  firmware has no handler for this (timer 3+4 and SFR D8h)
```

Blurb from AMT630A spec sheet: "Supports 13 standard interrupt sources include external interrupt, 3 Timer, watchdog etc"
Above four unknown vectors might include: I2C, GPIO, and whatever... maybe watchdog can be re-mapped to another vector than reset?

Open-bus areas in XRAM memory space

```
  3000h..FAFFh - Unused (open-bus) (CB00h bytes)
  FB8Ah..FBFFh - Unused (open-bus) (76h bytes)
  FC11h        - Unused (open-bus) (01h byte)
  FC47h..FC8Fh - Unused (open-bus) (49h bytes)
  FCB9h..FCBAh - Unused (open-bus) (02h bytes)
  FCEBh..FCFFh - Unused (open-bus) (15h bytes)
  FD60h..FDAFh - Unused (open-bus) (50h bytes)
  FDE8h..FDEFh - Unused (open-bus) (08h bytes)
  FDF2h..FDFFh - Unused (open-bus) (0Eh bytes)
  FEFFh        - Unused (open-bus) (01h byte)
  FFA4h..FFAFh - Unused (open-bus) (0Ch bytes)
  FFDDh        - Unused (open-bus) (01h byte)
  FFEBh..FFEFh - Unused (open-bus) (05h bytes)
  FFFCh..FFFFh - Unused (open-bus) (04h bytes)
```

Reading from open-bus areas does usually return FFh, or most recent value being read from FB00h..FFFFh region (or a mixup thereof, ie. some "0" bits from the recent value already faded to "1" bits).

AMT630A - SFRs - System Timers/Ports/etc

SFR 81h - SFR_CPU_sp
SFR 82h - SFR_CPU_dpl ;lsb of 16bit dptr
SFR 83h - SFR_CPU_dph ;msb of 16bit dptr
SFR D0h - SFR_CPU_psw
SFR E0h - SFR_CPU_a
SFR F0h - SFR_CPU_b
Standard 80C31 CPU registers.

SFR 80h - SFR_IO_PORT0_DATA (p0)
SFR 90h - SFR_IO_PORT1_DATA (p1)
SFR A0h - SFR_IO_PORT2_DATA (p2)
SFR B0h - SFR_IO_PORT3_DATA (p3)
Standard 80C31 peripheral I/O ports. Most of the ports are used for LCD video, some for the SPI FLASH memory system, and some are available for general purpose, and/or can be switched to special PWM/ADC/REMOTE/I2C modes, the mode selection for each pin is to be done via special SFR_IO_PORTx_MODE_A/B and IO_PIN_xxx registers.

SFR 87h - SFR_IO_PCON ;bit0-1:DANGER(halt/idle), bit4-5:NOT R/W(1?), bit6=?
SFR 88h - SFR_IO_TCON ;bit3:NOT R/W
SFR 89h - SFR_IO_TMOD
SFR 8Ah - SFR_IO_timer0_lsb (tl0)
SFR 8Bh - SFR_IO_timer1_lsb (tl1)
SFR 8Ch - SFR_IO_timer0_msb (th0)
SFR 8Dh - SFR_IO_timer1_msb (th1)
SFR 98h - SFR_IO_sio_scon ;serial UART control
SFR 99h - SFR_IO_sio_sbuf ;serial UART data ;read (R) and write (W)
SFR A8h - SFR_IO_iec ;extended, with extra bits in bit5-6 (set to 28h/A8h)
SFR B8h - SFR_IO_ipc ;extended, with extra bits in bit5-6, bit7:NOT R/W
Standard 80C31 control/timer/uart registers. Some register bits are slightly customized:

```
  PCON.4  unknown (read-only, always 1) ;maybe whatever 80C52-style POF bit?
  PCON.5  unknown (read-only, always 1)
  PCON.6  unknown (read/write-able)
  IEC.5 and IPC.5  probably 80C52-style Timer2 interrupt (read/write-able)
  IEC.6 and IPC.6  unknown... some extra interrupt?      (read/write-able)
```

SFR C8h - SFR_IO_timer2_control ;80C52-style extension
SFR CAh - SFR_IO_timer2_reloadcapture_lsb ;80C52-style extension
SFR CBh - SFR_IO_timer2_reloadcapture_msb ;80C52-style extension
SFR CCh - SFR_IO_timer2_counter_lsb (R) ;80C52-style extension
SFR CDh - SFR_IO_timer2_counter_msb (R) ;80C52-style extension

These Timer2 registers are a fairly common 80C52-based SFR extension. The firmware tries to initialize these to 57600 baud (but with improper rounding and insane div32 prescaler), alongsides it does enable Timer2 IRQs (although not having a proper irq vector for it, not having implemented any serial get/send byte functions).

```
____________________ AMT630A-specific Non-standard SFR's ____________________
```

SFR E8h - SFR_IO_IEC2 ;Interrupt Enable Flags

```
  0     Enable IRQ 0043h (adc)
  1     Enable IRQ 004Bh
  2     Enable IRQ 0053h
  3     Enable IRQ 005Bh (vblank/vsync or so)
  4     Enable IRQ 0063h (timer3+timer4)
  5-7   Unknown, not R/W (usually/always 111b)
```

Firmware initalizes 4bits in this register (and leaves 5th bit unchanged).
Note: This register is standard in so far as sharing the same SFR address as in P8xCE558 chips (but with different irq sources in the registers).

SFR F8h - SFR_IO_xxxF8h

```
  0-4   Seems to be related to SFR E8h.bit0-4, see there
  5-7   Unknown, not R/W (usually/always 111b)
```

Firmware initalizes 4bits in this register (and leaves 5th bit unchanged).
Maybe Interrupt PRIORITY bits?
Note: This register is standard in so far as sharing the same SFR address as in P8xCE558 chips (but with different irq sources in the registers).

SFR 91h - SFR_IO_xxx91h (usually 88h when reading?)

Extra Interrupt FLAGS (and acknowledge).

```
  0-3   Unknown, NOT R/W
  4     Write 0 for IRQ 0043h acknowledge (ADC) ;also need to ack IO_ADC_status
  5     Write 0 for IRQ 004Bh acknowledge
  6     Write 0 for IRQ 0053h acknowledge
  7     Write 0 for IRQ 005Bh acknowledge (vblank/vsync or so)
```

Oddly, bit4-7 are fully R/W, making it impossible to clear single bits without destroying the other bits (unless "AND [91h],mask" should happen to leave the unchanged bits untouched, instead of acting as RMW, which would cause any transitions between read & modify to get lost). Best might be to IGNORE the value in SFR 91h, and just write write 00h (clear all bits), and check for secondary IRQ flags in ADC/etc registers, instead of relying on the SFR 91h bits (though unknown if there's any such flag corresponding to bit7).
Bit7 can be acknowledged via SFR 91h.bit7 (without needing to ack any other registers). Bit7 gets set once per frame (at vsync/vblank or so). The framerate depends on the AV signal (50Hz/60Hz), if there's no AV signal then bit7 is kept thrown at the most recent AV framerate (with OSD frames being kept drawn at that framerate). Bit7 won't get set if the display PLL's are powered off.

Blurb from AMT630A spec sheet: "Supports 13 standard interrupt sources include external interrupt, 3 Timer, watchdog etc"

```
  There seem to be 7 interrupt sources in IEC/IPC
  There seem to be 5 interrupt sources in E8h/F8h
  And Reset/Watchdog might be the 13th interrupt source.
  However, there seem to be 14 exception vectors in total (including reset).
```

SFR C6h - SFR_IO_memory_system ;DANGER ;flash/osd memory system?

```
  0     DANGER: causes reboot when changed
  1     Unknown, NOT R/W
  2     Unknown
  3     Disable OSD TEXT rendering (should be set during FLASH-to-FONT DMA)
  4-6   Unknown
  7     Firmware sets bit7 upon power-up, DANGER: hangs when changed
```

SFR 8Eh - SFR_IO_whatever_config ;-lower 3bits set

Unknown, firmware sets the lower 3 bits during init (and leaves the upper 5bit unchanged).

SFR D8h - whatever ;bit0-2,6:NOT R/W ;bit5: makes CPU 21.25x slower

```
  0-2   Unknown, not R/W                                       ;not R/W
  3     Unknown, somehow forces IRQ 0063h (timer3+4)?          ;R/W
  4     Unknown                                                ;R/W
  5     Unknown, once seemed to make CPU 21.25x slower?        ;R/W
  6     Unknown, not R/W                                       ;not R/W
  7     Unknown                                                ;R/W
```

Whatever, parts R/W, not used by firmware.

SFR 92h - SFR_IO_xram_bank

Upper 8bit of XRAM address for 8bit addressing via "movx [r0]" (eg. allows to access 2K XRAM at 0000h-07FFh, and I/O ports at FB00-FFFFh). Not used by firmware (the firmware is doing all XRAM addressing via 16bit dptr).

SFR B1h - SFR_IO_timer3_lsb ;\faster timer (incrementing)
SFR B2h - SFR_IO_timer3_msb ;/
SFR B3h - SFR_IO_timer4_lsb ;\slower timer (4x slower than above)
SFR B4h - SFR_IO_timer4_msb ;/

```
  0-15  Incrementing Timer value (R/W)
```

Unknown if there's a way to stop the timers, or to change their speed.

```
  Timer 3 is faster
  Timer 4 is slower (4x slower than Timer 3)
```

SFR B5h - SFR_IO_timer34_stat ;NOT R/W ;03h

Reading returns overflow flags:

```
  0     Timer 3 overflow flag, not R/W (1=overflow occurred)
  1     Timer 4 overflow flag, not R/W (1=overflow occurred)
  2-7   Unknown/unused, not R/W
```

Writing acknowledges bits, oddly using bit1-2 instead of bit0-1:

```
  0     Unknown/unused
  1     Acknowledge Timer 3 overflow (0=No change, 1=Clear overflow flag)
  2     Acknowledge Timer 4 overflow (0=No change, 1=Clear overflow flag)
  3-7   Unknown/unused
```

Not used by firmware. Note: The overflow flags get set even when IRQs are disabled via SFR B6h and/or SFR E8h.

SFR B6h - SFR_IO_timer34_ctrl ;bit2-7:NOT R/W

```
  0     Timer 3 IRQ Enable (when master enable in SFR E8h.bit4 is set)
  1     Timer 4 IRQ Enable (when master enable in SFR E8h.bit4 is set)
  2-7   Unknown/unused, not R/W
```

Not used by firmware. Enables IRQ 0063h (used for both Timer 3+4).

SFR B9h - SFR_IO_watchdog_config1 ;when locked: NOT R/W ;03 ;bit2-7:never R/W
SFR BAh - SFR_IO_watchdog_enable ;when locked: NOT R/W ;00 ;bit1-7:never R/W
SFR BBh - SFR_IO_watchdog_reload ;when locked: NOT R/W ;00 ;bit1-7:never R/W
SFR BCh - SFR_IO_watchdog_config2 ;when locked: NOT R/W ;FF ;fully R/W
SFR BDh - SFR_IO_watchdog_config3 ;when locked: NOT R/W ;8F ;fully R/W
SFR BEh - SFR_IO_watchdog_unlock ;when locked: NOT R/W ;00 ;never R/W?

When enabled, resets the CPU when not reloading it for about 1 second.

SFR E9h - SFR_IO_PORT0_MODE_A ;bit7,6=PWM, bit0,1,2=ADC, bit3=IR
SFR EDh - SFR_IO_PORT0_MODE_B ;bit7,6=PWM, bit0,1,2=ADC, bit3=IR
SFR EAh - SFR_IO_PORT1_MODE_A ;(?)
SFR EEh - SFR_IO_PORT1_MODE_B ;(?)
SFR EBh - SFR_IO_PORT2_MODE_A ;(?)
SFR EFh - SFR_IO_PORT2_MODE_B ;(?)
SFR ECh - SFR_IO_PORT3_MODE_A ;bit7,6,5=PWM
SFR F4h - SFR_IO_PORT3_MODE_B ;bit7,6,5=PWM

Used to configure Port0-3, combined with the 4bit IO_PIN_xxx settings, following combinations are known:

```
   A  B  4bit
   0  1  x000    Digital.output (eg. for LCD SPI writes, or static backlight)
   1  0  x000    Digital.input  (eg. for LCD SPI reads)
   1  1  xxxx    ADC (analog/digital converter, eg. keypad button read)
   x  x  xx11    PWM (pulse-width output, eg. for dimmed backlight)
   x  x  0001    LCD 3.5" (RGB+CLK+DEN+SYNC) and SPI FLASH pins
   x  x  0002    LCD 4.3" (RGB+CLK+DEN)
   1  0  xxxx    REMOTE (infrared IR input)
```

SFR E4h - SFR_IO_IR_data ;NOT R/W ;FFh (Write FFh to acknowledge IRQ?)

```
  0-7   Command from REMOTE (further stuff is in FFxxh register area)
```

SFR E5h - SFR_IO_IR_flags ;NOT R/W ;00h (Write FEh to acknowledge bit0?)

```
  0     Flag (maybe new command?)
  1     Flag (maybe repeat?)
  2     Unknown (usually 0, but I think've seen the register being 04h once)
  3-7   Unknown (usually 0)
```

SFR 86h - whatever ;bit0:DANGER, bit1-7:NOT RW
SFR 8Fh - whatever ;bit0:SKIPPED???(force ADC=ready?), bit1-7:NOT R/W
SFR F5h - whatever ;bit1-7:NOT R/W
Whatever, parts R/W, not used by firmware.

SFR E1h - whatever status? ;NOT R/W ;FFh ;
SFR E2h - whatever status? ;NOT R/W ;FFh ;
SFR E3h - whatever status? ;NOT R/W ;FFh ;/
Whatever, seems to be read-only, not used by firmware.

SFR 84h..85h - whatever, 2 bytes (usually 00h, but read/write-able)
SFR A1h..A7h - whatever, 7 bytes (usually 00h, but read/write-able)
SFR A9h..AFh - whatever, 7 bytes (usually 00h, but read/write-able)
SFR D2h..D5h - whatever, 4 bytes (usually 00h, but read/write-able)
SFR C7h - whatever, 1 byte (usually 00h, but read/write-able)
Whatever, 8bit fully R/W, not used by firmware.

SFR 93h..97h - unknown/unused, 5 bytes (not R/W, returns 00h on reading)
SFR 9Ah..9Fh - unknown/unused, 6 bytes (not R/W, returns 00h on reading)
SFR C0h..C5h - unknown/unused, 6 bytes (not R/W, returns 00h on reading)
SFR CEh..CFh - unknown/unused, 2 bytes (not R/W, returns 00h on reading)
SFR D6h..D7h - unknown/unused, 2 bytes (not R/W, returns 00h on reading)
SFR D9h..DFh - unknown/unused, 7 bytes (not R/W, returns 00h on reading)
SFR E6h..E7h - unknown/unused, 2 bytes (not R/W, returns 00h on reading)
SFR F1h..F3h - unknown/unused, 3 bytes (not R/W, returns 00h on reading)
SFR F6h..F7h - unknown/unused, 2 bytes (not R/W, returns 00h on reading)
SFR F9h..FFh - unknown/unused, 7 bytes (not R/W, returns 00h on reading)
SFR B7h - unknown/unused, 1 byte (not R/W, returns 00h on reading)
SFR BFh - unknown/unused, 1 byte (not R/W, returns 00h on reading)
SFR C9h - unknown/unused, 1 byte (not R/W, returns 00h on reading)
SFR D1h - unknown/unused, 1 byte (not R/W, returns 00h on reading)
These registers seem to be always 00h, not read/write-able. Maybe they are just unused SFR addresses.

AMT630A - FBxxh - OSD Registers (On-Screen Display)

```
  ________________________________ Window 0..4  ________________________________
```

FB07h/FB12h/FB18h/FB1Eh/FB24h - IO_OSD_window_N_size_x (N=0..4)

```
  0-6  Horizontal Window Size in Characters (1..127)
  7    Not used (always 0)
```

FB08h/FB13h/FB19h/FB1Fh/FB25h - IO_OSD_window_N_size_y (N=0..4)

```
  0-5  Vertical Window Size in Characters (1..63)
  6-7  Not used (always 0)
```

FB09h/FB14h/FB1Ah/FB20h/FB26h - IO_OSD_window_N_xyloc_msb (N=0..4)

```
  0-2  Upper 3bit of 11bit Horizontal Window position
  3    ???
  4-6  Upper 3bit of 11bit Vertical Window position
  7    Upper 1bit of 9bit Window's BGMAP Address ;Window 0: Not used (always 0)
```

FB0Ah/FB15h/FB1Bh/FB21h/FB27h - IO_OSD_window_N_xloc_lsb (N=0..4)

```
  0-7  Lower 8bit of 11bit Horizontal Window position (0..7FFh) (10=leftmost)
```

FB0Bh/FB16h/FB1Ch/FB22h/FB28h - IO_OSD_window_N_yloc_lsb (N=0..4)

```
  0-7  Lower 8bit of 11bit Vertical Window position (0..7FFh) (12=topmost)
```

FB17h/FB1Dh/FB23h/FB29h - IO_OSD_window_N_vramaddr_lsb (N=1..4 only, not N=0)

```
  0-7  Lower 8bit of 9bit Window's BGMAP Address (0..1FFh)
```

Note: Address for Window 0 is fixed (always 000h).

The xloc/yloc values are unsigned 0..7FFh (ie. 7FFh will hide the OSD window, rather than being treated as "-1"). Nethertheless parts offscreen tiles at upper/left screen edges are possible with xloc=0..9 or yloc=0..11 (that, depending on the OSD position defined in FCxxh registers).

```
  ______________________________ Control Regs ______________________________
```

FB05h - IO_OSD_window_enable_bits

```
  0-4   Enable Window 0..4  (0=Off, 1=On)
  5     ???
  6     Disable 1bpp Text       (0=Normal, 1=Force all 1bpp Tiles Black)
  7     Enable 4bpp Bitmap      (0=Off, 1=4bpp for IO_OSD_bitmap_start and up)
```

FB06h - IO_OSD_misc_transp_enable

```
  0-5   ???
  6     SemiTransparency Enable for BG   (0=Solid, 1=SemiTransp)
  7     SemiTransparency Enable for TEXT (0=Solid, 1=SemiTransp, if bit6=1)
```

FB0Ch - IO_OSD_bright_transp_level
```
  0-2   SemiTransparency (0=OsdIsInvisible, 1..7=more and more opaque)
  3-4   ???
  5-7   Brightness (0=Black, 1=Dark, 2=Med, 3=Bright, 4=Max/Normal, 5..7=Crop)
```
SemiTransparency makes the AV layer visible even underneath of OSD pixels, the effect can be enabled in FB06h for BG and TEXT.
Brightness adjust works as "RGB=RGB*Brightness(0..7)/4", and crops the result to max 0Fh. The firmware does (maybe accidentally) use Brightness=7, resulting in only 11 different RGB intensities (00h..0Ah, and 0Bh..0Fh all appearing as dupes of 0Ah).

FB78h - IO_OSD_xyflip (used by unused firmware functions)
```
  0-3   ???
  4     Xflip Tiles (0=Normal; 1st Pixel (MSB) is left, 1=Mirror Horizontally)
  5     Xflip BGMAP (0=Normal; 1st Tile is left,        1=Mirror Horizontally)
  6     Yflip Tiles (0=Normal; 1st Pixel-row is upper,  1=Mirror Vertically)
  7     Yflip BGMAP (0=Normal; 1st Tile-row is upper,   1=Mirror Vertically)
```
Note: The window positions must be manually adjusted for flipping, eg.
```
  xloc = osd_base_x + xloc                                ;\normal
  yloc = osd_base_y + yloc                                ;/
  xloc = osd_base_x + screen_width - window_width - xloc   ;\flipped
  yloc = osd_base_y + screen_height - window_height - yloc ;/
```
Flipping affects OSD only. There's no known way to flip AV at AMT630A side (though maybe it does support xflip somehow, yflip is probably not possible due to limited line buffer size). However, some LCD displays are supporting x/y-flipping via SPI bus commands (eg. NV3035C-based Tianma displays, or HX8238-D-based Noname displays, whilst Innolux displays aren't having SPI bus at all). For example, one could apply xflip to OSD and also to the whole LCD image - resulting in mirrored AV image with normal OSD output (due to double flipping applied to OSD). Moreover, if AV comes from a camera: some cameras might support flipping.

FB35h - IO_OSD_bitmap_transp_misc (set to 00h by firmware)
```
  0-2   ???
  3     Affects Horizontal Window positions (0=Normal, 1=Shift SOME pixels)
  4     Bitmap Color 0 (aka 4bpp tiles)     (0=Transparent, 1=Solid)
  5     Affects Vertical Window positions   (0=Normal, 1=Shift ONE pixel)
  6-7   Not used (always 0)
```

FB62h - IO_OSD_whatever_FB62h (bit0 cleared by firmware) (no visible effect)
```
  0     ??? (cleared by firmware; alongsides when disabling the 5 windows)
  1-7   ???
```

FB89h - IO_OSD_screen_position (set to 00h by firmware) (window positions?)
```
  0     Move windows 2pix DOWN, and SOME pix LEFT
  1     Jitters! Scanline 1pix shift (with AV: each 2nd line, no AV: each 3rd)
  2     ???
  3     Move windows SOME pix DOWN (bottom-most pixels wrap to top of screen)
  4-7   Not used (always 0)
```

Bit1 moves all windows SOME pix RIGHT, and also each 2nd/3rd scanline one more pixel left/right (without AV that's each 3rd line and it's fixed in all frames, with AV it's each 2nd line and it's alternating for odd/even lines in odd/even frames).

```
  ________________________________ Window Scale ________________________________
```

FB2Bh - IO_OSD_window_0_vscale_lsb_upper_8_pixels
FB2Ch - IO_OSD_window_0_vscale_mid_middle_8_pixels (if height>8)
FB2Dh - IO_OSD_window_0_vscale_mid_lower_8_pixels (if height>16)
FB2Eh - IO_OSD_window_0_vscale_msb_lowest_8_pixels (if height>23)

```
  0-31  Scale 1st..32th pixel within Font tile (bit0=topmost)  (0=No, 1=Scale)
```

FB2Fh - IO_OSD_window_0_hscale_lsb_left_8_pixels
FB30h - IO_OSD_window_0_hscale_mid_middle_8_pixels (if width>8)
FB31h - IO_OSD_window_0_hscale_msb_right_8_pixels (if width>16)

```
  0-23  Scale 1st..24th pixel within Font tile (bit0=leftmost) (0=No, 1=Scale)
```

FB32h - IO_OSD_window_0_scale

```
  0-1  Window 0 Horizontal Scale (0=Double, 1=Triple, 2=Quad, 3=FiveX)
  2-3  Window 0 Vertical Scale   (0=Double, 1=Triple, 2=Quad, 3=FiveX)
  4-7  Not used (always 0)
```

For window 0, scaling is applied only to the pixels selected in hscale/vscale bits. Normally the vscale/hscale bits would be set to all 00h's or all FFh's. Other settings are resulting in smeared appearance with some pixels being wider than others. However, that effect may be useful in a few cases: For animated zoom-in/out effects, for stretching the character spacing (when lower/right pixels are all blank), or for stretching certain sections of uniformly defined symbols (eg. the 2nd/4th/6th lines of uppercase letters with 6pix height).

FB33h - IO_OSD_window_1_and_2_scale

```
  0-1  Window 1 Horizontal Scale (0=Normal, 1=Double, 2=Triple, 3=Quad)
  2-3  Window 1 Vertical Scale   (0=Normal, 1=Double, 2=Triple, 3=Quad)
  4-5  Window 2 Horizontal Scale (0=Normal, 1=Double, 2=Triple, 3=Quad)
  6-7  Window 2 Vertical Scale   (0=Normal, 1=Double, 2=Triple, 3=Quad)
```

FB34h - IO_OSD_window_3_and_4_scale

```
  0-1  Window 3 Horizontal Scale (0=Normal, 1=Double, 2=Triple, 3=Quad)
  2-3  Window 3 Vertical Scale   (0=Normal, 1=Double, 2=Triple, 3=Quad)
  4-5  Window 4 Horizontal Scale (0=Normal, 1=Double, 2=Triple, 3=Quad)
  6-7  Window 4 Vertical Scale   (0=Normal, 1=Double, 2=Triple, 3=Quad)
```

```
  ______________________________ Window Palette ______________________________
```

FB36h+(#*2) - IO_OSD_bitmap_color_#_msb_B (#=0..15) (for 4bpp)
FB37h+(#*2) - IO_OSD_bitmap_color_#_lsb_GR (#=0..15) (for 4bpp)
FB56h/FB58h/FB5Ah/FB5Ch/FB5Eh/FB60h - IO_OSD_color_#_msb_B (#=1..6)
FB57h/FB59h/FB5Bh/FB5Dh/FB5Fh/FB61h - IO_OSD_color_#_lsb_GR (#=1..6)

N/A - IO_OSD_color_0: color 0 is fixed (transparent, aka AV video)
N/A - IO_OSD_color_7: color 7 is fixed (black)

```
  0-3   Red      (0-10)  ;(11-15 are SAME as 10) ;\LSB (second byte) (!)
  4-7   Green    (0-10)  ;(11-15 are SAME as 10) ;/
  8-11  Blue     (0-10)  ;(11-15 are SAME as 10) ;\MSB (first byte) (!)
  12-15 Not used (always 0)                      ;/
```

Intensities 0Ah..0Fh are all max brightness (with default gamma setting, 08h/09h are also looking nearly identical to max brightness). At least that's so at default configuration - not tested what happens when dividing the intensities (via bright/semitransp features), maybe that results in cases like 0Fh/2 being brighter than 0Ch/2.

Transparent Color & AV Video/Backdrop & Overlapping Windows
For 1bpp characters, color 0 is always transparent. For 4bpp characters, color 0 transparency is optional (see port FB35h).
Transparent means displaying the AV Video Layer (or the backdrop if no AV signal is present; the appearance of the backdrop can be changed via I/O ports in the AV engine; eg. blue, black or other color, with or without snow effect).
In case of overlapping windows, window 0 is having highest priority. However, any transparent pixels in the frontmost window are simply showing AV/Backdrop (instead of drawing pixels from underlaying windows).
Moreover, half-overlapping windows are glitchy when their positions aren't equally aligned to the font character width. Glitches are getting worse if the overlapping windows are each using different scaling settings.

```
  ________________________________ BGMAP Memory ________________________________
```

FB00h - IO_OSD_bgmap_addr_lsb (lsb with internal auto-incrementing lsb)
FB0Dh - IO_OSD_bgmap_addr_msb (msb fixed, need manual increment for msb)

```
  0-7   Lower 8bit of 9bit BGMAP Address (0..1FFh)      ;-LSB
  8     Upper 1bit of 9bit BGMAP Address                ;\MSB
  9-15  Not used (always 0)                             ;/
```

FB01h - IO_OSD_bgmap_data_lsb
FB0Eh - IO_OSD_bgmap_data_msb

```
  0-7   Lower 8bit of 10bit Character Number (0..3FFh)  ;-LSB
  8-9   Upper 2bit of 10bit Character Number (0..3FFh)  ;\MSB
  10-15 Not used (always 0)                             ;/
```

FB10h - IO_OSD_bgmap_data_attr (for 1bpp tiles)

```
  0-2   Text Color       (0=Transparent, 1..6=Variable, 7=Black)
  3     ???
  4-6   Background Color (0=Transparent, 1..6=Variable, 7=Black)
  7     Not used (always 0)
```

BGMAP Memory

```
  200h entries (each entry is 10bit character number plus 7bit attribute)
```

Character Numbers

```
  000h..1BFh    Fixed ROM Font  (1bpp, with color attributes)
  1C0h..xxxh    Custom RAM Font (1bpp, with color attributes)
  xxxh..xxxh    Custom RAM Font (4bpp, aka "16-color bitmap")
  xxxh..3FFh    Not useable (unless char height<8) (due to font ram limit)
```

```
  _______________________________ FONT Memory ________________________________
```

FB02h - IO_OSD_font_addr_lsb
FB0Fh - IO_OSD_font_addr_msb

```
  0-7   Lower 8bit of 12bit Font Address (0..FFFh)      ;-LSB
  8-11  Upper 4bit of 12bit Font Address                ;\MSB
  12-15 Not used (always 0)                             ;/
```

Used for manual font writes (not used when DMA uploading FLASH-to-FONT).

FB03h - IO_OSD_font_data_lsb
FB04h - IO_OSD_font_data_msb

```
  0-7   Lower 8bit of 16bit Word (bit0=leftmost pixel)  ;-LSB
  8-15  Upper 8bit of 16bit Word                        ;-MSB
```

Used for manual font writes (not used when DMA uploading FLASH-to-FONT).

FB76h - IO_OSD_char_xsiz

```
  0-4   Horizontal Character Size in pixels (1..24) (25..32=glitchy) (0=32)
  5-7   Not used (always 0)
```

FB77h - IO_OSD_char_ysiz

```
  0-5   Vertical Character Size in pixels (1..32) (33..63=glitchy) (0=freeze)
  6-7   Not used (always 0)
```

FB11h - IO_OSD_bitmap_start_lsb
FB70h - IO_OSD_bitmap_start_msb

Defines the first character number that is to be drawn at 4bpp color depth (provided that 4bpp is enabled via FB05h.bit7). The value is originated at character 1C0h (the first custom character).

```
  0-7   Lower 8bit of 10bit Character Number (0..(3FFh-1C0h))  ;-LSB
  8-9   Upper 2bit of 10bit Character Number (0..(3FFh-1C0h))  ;\MSB
  10-15 Not used (always 0)                                    ;/
```

The actual first useable 4bpp character number varies depending on the character width (because 4bpp char numbers are always addressing font memory as wordaddr=char*ysiz). For example, using start=40h, and width<=16:

```
  1bpp characters = 1C0h..1FFh (40h characters)
  4bpp characters = 200h and up
```

However, at width>16 some 4bpp chars will overlap the 1bpp font memory area:

```
  1bpp characters = 1C0h..1FFh (40h characters)
  4bpp garbage    = 200h..21Fh (20h characters)
  4bpp characters = 220h and up
```

Note: The math for width>16 gets yet more complex when char.ysiz=odd (when 1bpp tiles include a half unused word).

OSD ROM Font

The ROM font is 16x22 pixel, 1bpp. Smaller character sizes are causing cropped ROM symbols (showing only the upper/left pixels). Bigger character sizes are inserting additional character spacing. However height 32..63 is wrapping within the tile bitmap (causing the rows to be displayed twice).

```
  000h        Space
  001h..00Ah  Digits "0..9"
  00Bh..024h  Letters "A..Z" (uppercase only)
  025h..02Bh  Letters "ZSCNLZE" (with accent marks)
  02Ch..04Ch  Cyrillic Letters (in uncommon order)
  02Dh..030h  Symbols "<", ">", Arrow/TriangleLeft, Arrow/TriangleRight
  031h..032h  Symbols for Enter (left, right half)
  033h..037h  Symbols for Striped Bar (with 0,1,2,3,4 bars)
  038h..039h  Symbols for Double Arrows (Up+Down, and Right+Left)
  03Ah..040h  Symbols for Solid Bar (LeftEdge, 0,1,2,3,4 bars, RightEdge)
  041h..042h  Symbols for Sound or so (SpeakerX and "o))")
  043h..0D3h  Japanese and/or chinese or so
  0D4h..0D5h  Symbols for Alert (left, right half)
  0D6h..105h  Japanese and/or chinese and/or whatever or so
  106h..10Bh  Symbols for Contrast, Brightness, Palette (left, right halves)
  10Ch        "."
  10Dh        "'C"
  10Eh..131h  Japanese and/or chinese and/or whatever or so
  132h..133h  Symbol for HandWithFingerRight and ZigZagParagraph
  134h        ":"
  135h        "!"
  136h        "?"
  137h..13Bh  Symbols for Battery (0,1,2,3 bars, and right half)
  13Ch..15Ch  Letters "AAAAAAEEEEUUUUIIIIIOOOOOODNGZYYap" (with accent marks)
  15Dh..15Fh  Symbols for upper border ".---."
  160h..164h  Letters "zShoC" (with accent marks/variants)
  165h..17Ah  Japanese and/or chinese and/or dental or so
  17Ch..17Dh  Symbols for left/right border "|" and "|"
  17Eh..197h  Japanese and/or chinese and/or dental or so
  198h..19Ah  Symbols for lower border "'---'"
  19Bh..19Eh  Japanese and/or chinese and/or dental or so
  19Fh        Symbol for ClockfaceTrifoldYingYang or so
  1A0h        Overscore
  1A1h        Underscore "_" with clean line
  1A2h        Underscore "_" with noisy line
  1A3h..1BFh  Unused (solid 16x22pix rectangle's)
```

Custom Font Memory (RAM)

```
  1000h words (aka 2000h bytes, aka 8 Kbytes)
```

1bpp font RAM addressing

For 1bpp font RAM addressing, the width is rounded up to 16bits (when char.width<=16), or to 24bits (when char.width>16).
The size of each tile is Width (=rounded to 16bit or 24bit), multiplied by the Height (variable size). The result is then rounded up to a multiple of 16bits (ie. there'll be a trailing dummy byte in case of 24bit width with odd height).
The 1bpp tile bitmaps always consist of straight 16pix rows (plus patchwork for width>16). The pixels arranged as so:

```
  1-8      9-16     17-24         25-31   32      ;<-- pixels
  msb(0)   lsb(0)   msb(ysiz+0)   blank   buggy   ;1st row
  msb(1)   lsb(1)   lsb(ysiz+0)   blank   buggy   ;2nd row
  msb(2)   lsb(3)   msb(ysiz+1)   blank   buggy   ;3rd row
  msb(3)   lsb(3)   lsb(ysiz+1)   blank   buggy   ;etc.
```

4bpp font RAM addressing

4bpp tileno's are simply addressing memory as "wordaddr=tileno*char.height" (regardless of char.width). The number of words per 4bpp row is "N=(char.width+3)/4", so one should use only each N'th tileno for 4bpp tiles (unless one wants to use overlapping tiles).
The 4bpp tile bitmaps consist of N*4pix rows. The pixels arranged as so:

```
  1-4       5-8       9-12      13-16     17-20     21-24     25-28     29-32
  msb0 lsb0 msb1 lsb1 msb2 lsb2 msb3 lsb3 msb4 lsb4 msb5 lsb5 msb6 lsb6 buggy
```

The next row starts with next msb (eg. with msb5 for width=17..20).

1bpp font ROM addressing

The tileno's for the ROM font are always addressing the corresponding ROM character regardless of the current character width/height (showing only the upper/left pixels when width/height is less than 16x22, and adding extra blank space when above 16x22).

Font Write Glitches

Writes to FONT memory can be glitchy (if the write occurs simultaneously when rendering a tile; statistically that's most likely for 4bpp with 1pix width, which have the most vram traffic, but the glitch seems to happen at other width's sometimes too).
Workaround would be to disable all windows before FONT writes (and use some DELAY after window-disable, which is apparently taking no effect until next frame or so) (the SFR_IO_memory_system flag for DMA-FONT writes does NOT work for manual FONT writes, instead, it does totally screw up such writes).
Unknown if font writes are more reliable during vblank (see irq flag in SFR 91h).

```
  _________________________BGMAP+FONT Memory Notes _________________________
```

Data Write & Address Increment

The "addr_lsb" registers are auto-incrementing when writing to "data_lsb" (or "data_attr"). However, there is no carry-out to "addr_msb", so the msb must be written manually upon lsb-overflows. Another oddity is that reading "addr_lsb" does return the must recently written value (not the internally auto-incremented value; for bgmap, there are separate internal auto-incrementing addr_lsb's for data and attr writes, so the auto-increments don't disturb each other).
Writes to "data_msb" register are just latching the written value (without forwarding it to vram). Writes to "data_lsb" are forwarding the LSB and MSB to vram.
When zero-filling memory, one needs to write MSB only once, and can then write LSB several times. For fonts with 8pix width, the pixels are located in MSB only (but one still needs dummy LSB writes to get the MSBs forwared to vram).

Memory Restrictions

The OSD is focusing on "on-screen-display", but it's a bit short of memory for "full-screen display" purposes. Making use of the whole Font, Bgmap memory and all character numbers works only with well-balanced character sizes (for example, 16x8 pixels).
Nethertheless, with a 320x240 pixel display, the following is allowing to display a near-fullscreen bitmap, using nearly all memory:

```
  20x25 tiles bgmap, 16x8 pixel tiles at 1bpp = 320x200 pixels
```

when manually drawing text pixels, the above could be also used for stuff like 40x25 tiles at 8x8 pixels (though sharing the same color attribute for each two characters).
The 16-color mode requires 4x more font memory per pixel, thus allowing use only about of a quarter of the screen (unless one would scale a low-res 160x100 pixel bitmap to a 320x200 pixel region).

Absent Memory Read Support

There's no known way to read bgmap/font memory (trying to read the data registers merely returns the most recently written value).
The absent read-support can be problematic for bitmap graphics (drawing a pixel will destroy the other pixels within the same word, unless you still know the value of the old pixels).
Unless... maybe there are separate "data-read" registers somewhere, or some kind of "strobe to read" registers?

DMA Transfers

Apart from manually writing font data, one can also DMA transfer data from SPI/FLASH to FONT memory. Doing so is slightly faster, and allows to use higher FLASH addresses (exceeding the MOVC opcode's 16bit address space).

```
  ____________________________ Unknown Registers ____________________________
```

Below registers are unknown and seem to have no visible effect. Maybe some of them are for the "OSD blink/highlight" feature which is said to be supported "for THREE windows"... and which might become visible only if certain window(s) are displayed, with certain attribute bit(s), and certain blink-timing settings, and certain blink-enable flags... or whatever.

FB2Ah - Unknown (not used by firmware)

```
  FB2Ah - bit7:NOT R/W
```

FB63h-FB6Fh - Unknown (not used by firmware)

```
  FB63h - fully R/W
  FB64h - fully R/W
  FB65h - fully R/W
  FB66h - fully R/W
  FB67h - bit7:NOT R/W
  FB68h - fully R/W
  FB69h - fully R/W
  FB6Ah - bit7:NOT R/W
  FB6Bh - fully R/W
  FB6Ch - fully R/W
  FB6Dh - fully R/W
  FB6Eh - fully R/W
  FB6Fh - fully R/W
```

FB71h-FB75h - Unknown (not used by firmware)

```
  FB71h - bit1-7:NOT R/W
  FB72h - fully R/W
  FB73h - fully R/W
  FB74h - fully R/W
  FB75h - fully R/W
```

FB79h-FB88h - Unknown (not used by firmware)

```
  FB79h - bit6-7:NOT R/W
  FB7Ah - bit6-7:NOT R/W
  FB7Bh - bit7:NOT R/W
  FB7Ch - bit7:NOT R/W
  FB7Dh - bit4-7:NOT R/W
  FB7Eh - bit6-7:NOT R/W
  FB7Fh - bit6-7:NOT R/W
  FB80h - bit6-7:NOT R/W
  FB81h - bit7:NOT R/W
  FB82h - bit7:NOT R/W
  FB83h - bit4-7:NOT R/W
  FB84h - bit6-7:NOT R/W
  FB85h - bit6-7:NOT R/W
  FB86h - bit6-7:NOT R/W
  FB87h - bit7:NOT R/W
  FB88h - bit7:NOT R/W
```

FB8Ah..FBFFh - Unused (open-bus)
These registers appear to be unused (not write-able, always returning FFh on reading).

FBC6h - Unused (open-bus), or IO_OSD_bugged_color_lsb
FBC7h - Unused (open-bus), or IO_OSD_bugged_color_msb
For some reason (maybe by mistake), the firmware is writing some RGB color value to FBC6h/FBC7h, oddly done in LSB-first order (unlike the other RGB color registers), and more oddly with y-flipped tiles, but without visible color changes on the OSD picture, and without allowing to readback the written value (so FBC6h/FBC7h are either write-only, or, more likely, they don't exist at all).

AMT630A - FCxxh - LCD Registers (mostly 50Hz/60Hz/Ratio)

```
 ______________________ 60Hz/NTSC and 50Hz/PAL Registers ______________________
```

Registers below exist at separate addresses for 60Hz/NTSC and 50Hz/PAL,

```
  FC91h..FCB6h - IO_60HZ_xxx : used for 60Hz/NTSC (also used for 60Hz/PAL60)
  FCBDh..FCE2h - IO_50HZ_xxx : used for 50Hz/PAL (and probably 50Hz/SECAM)
```

FC91h/FCBDh - IO_60HZ/50HZ_control_lsb ;(should be set to 01h) ;PAL: bit4=?
FC92h/FCBEh - IO_60HZ/50HZ_control_mid ;(should be set to 00h) ;NTSC: bit12=?
FC93h/FCBFh - IO_60HZ/50HZ_control_msb ;(should be set to 00h)

The firmware initializes these registers to 24bit value 000001h (and does somewhat oddly change NTSC/bit12 and PAL/bit4 in some situation).

```
  1     Move Whole Image ca.80pix LEFT
  3     AV Black/Off
  5     Weird Diagonal/striped AV Image (ignoreHsync?),
  6     PAL: MoveWholeImageCa.12pixDOWN
  6     PAL60: ContrastOnPAL60colorError
  6     C64: shows SNOW with downwards rolling VSYNC from C64)
  8     AV Edgy/Diagonal Color Error,
  8     C64: shows SNOW with upwards rolling VSYNC from C64)
  11    PAL: AV diagonally messed (ignoredHsync?)
  11    PAL60: AV_MOSTLY_BLACK_except_right_edge_and_lowerleft_corner
  18    PAL: MoveWholeImageCa.160pixLEFT,
  18    PAL60: MoveWholeImageCa.80pixRIGHT,
  19    PAL: MoveWholeImageCa.32pixDOWNandWeakPixels
  19    PAL60: ContrastOnPAL60colorError
```

BUG: The firmware changes PAL/bit4 and NTSC/bit12 (done in response to bit0 of IO_AV_stat_detect_1), there might be some reason for changing different bits for PAL and NTSC, but probably it did want to change either bit4 or bit12 in both cases (neither of those bits has any visible effect though) (alongsides, it's also changing FCB6h/FCE2h, which is having a more visible effect on boldness).

FC98h/FCC4h - IO_60HZ/50HZ_15khz_lsb ;fixed (C8h/66h)
FC99h/FCC5h - IO_60HZ/50HZ_15khz_msb ;fixed (03h/04h)
FC9Ah/FCC6h - IO_60HZ/50HZ_15khz_div2_lsb ;fixed (E3h/36h)
FC9Bh/FCC7h - IO_60HZ/50HZ_15khz_div2_msb ;fixed (01h/02h)

These seem to somehow define the expected 15kHz scanline rate (which is slightly different for PAL and NTSC). The "div2" value should be approximately half of the other value. The firmware uses the following fixed values:

```
  60Hz/NTSC:  03C8h,01E3h (aka decimal 968,483)
  50Hz/PAL:   0466h,0236h (aka decimal 1126,566)
```

BUG: The 60Hz/NTSC value doesn't match for PAL60 (causing the display to forcefully insert memorized/old scanlines at random locations in bottom half of the screen; this can be fixed (eg. change 03C8h to 03C0h, or 01E3h to 01E2h; even then, the PAL60 colors are still decoded as NTSC though).

FC9Ch/FCC8h - IO_60HZ/50HZ_xloc_av_osd_lsb ;fixed (00h/02h)
FC9Dh/FCC9h - IO_60HZ/50HZ_xloc_av_osd_msb ;fixed (00h/00h)
FC9Eh/FCCAh - IO_60HZ/50HZ_xloc_osd_lsb ;fixed (44h/36h)
FC9Fh/FCCBh - IO_60HZ/50HZ_xloc_osd_msb ;fixed (00h/00h)
FCA0h/FCCCh - IO_60HZ/50HZ_xloc_av_lsb ;fixed (43h/43h)
FCA1h/FCCDh - IO_60HZ/50HZ_xloc_av_msb ;fixed (00h/00h)

FCACh/FCD8h - IO_60HZ/50HZ_ratio_xloc_av_8bit ;fixed (10h/06h)
Allows to change the horizontal position of both AV+OSD layer, or only AV layer, or only OSD layer. The firmware uses the following fixed values:
```
  60Hz/NTSC: both=0000h, osd=0044h, av=0043h, ratio=10h (or unused:1Dh)
  50Hz/PAL:  both=0002h, osd=0036h, av=0043h, ratio=06h (or unused:0Ch)
```
BUG: With the firmware values, the OSD layer appears at different horizontal locations depending on whether receiving a NTSC or PAL signal.
Note: Setting "av_osd" or "osd" to too large values causes TFT screen to freeze.
Note: Changing the "ratio" value may produce GARBAGE after the scanline end. The firmware contains tables with several used (unused) "ratio" values (used are 10h/06h, and whatever unused values would be 1Dh/0Ch).

FCA2h/FCCEh - IO_60HZ/50HZ_xcrop_end_av_lsb ;fixed (8Ah/92h)
FCA3h/FCCFh - IO_60HZ/50HZ_xcrop_end_av_msb ;fixed (01h/01h)
FCA8h/FCD4h - IO_60HZ/50HZ_ycrop_upper_av_lsb ;fixed (10h/0Ch)
FCA9h/FCD5h - IO_60HZ/50HZ_ycrop_upper_av_msb ;fixed (00h/00h)
FCAFh/FCDBh - IO_60HZ/50HZ_ratio_ycrop_upper_av_8bit ;fixed (00h)
FCB0h/FCDCh - IO_60HZ/50HZ_ratio_ycrop_lower_av_8bit ;fixed (00h)
Allows to crop the upper/lower/right areas of the AV layer (causing it to get black, OSD isn't affected). The firmware uses the following fixed values:
```
  60Hz/NTSC: x=018Ah, y=0010h, ratio/upper=00h, ratio/lower=00h
  50Hz/PAL:  x=0192h, y=000Ch, ratio/upper=00h, ratio/lower=00h
```
For the "ratio" values, the firmware contains tables for six ratios for PAL and NTSC each... but those tables are just containing 00h crop values for all ratios.

FCA4h/FCD0h - IO_60HZ/50HZ_yloc_av_osd_lsb ;fixed (07h/08h)
FCA5h/FCD1h - IO_60HZ/50HZ_yloc_av_osd_msb ;fixed (00h/00h)
FCA6h/FCD2h - IO_60HZ/50HZ_yloc_osd_lsb ;fixed (09h/09h)
FCA7h/FCD3h - IO_60HZ/50HZ_yloc_osd_msb ;fixed (00h/00h)
Allows to change the vertical position of AV+OSD layer, or only OSD layer. The firmware uses the following fixed values:
```
  60Hz/NTSC: both=0007h, osd=0009h
  50Hz/PAL:  both=0008h, osd=0009h
```
Note: Setting "av_osd" to too large values causes TFT screen to draw single scanline repeated all over the screen. Too large "av" value causes the AV layer to get offscreen (and AV area will be drawn as black).

FCAAh/FCD6h - IO_60HZ/50HZ_heavy_flimmer_lsb ;fixed (10h/0Fh)
FCABh/FCD7h - IO_60HZ/50HZ_heavy_flimmer_msb ;fixed (01h/01h)
Whatever, causes heavy flimmering when set to wrong values. The firmware uses the following fixed values:
```
  60Hz/NTSC: 0110h
  50Hz/PAL:  010Fh
```
Maybe forces vblank duration, large values cause screen to blink black.

FCB6h/FCE2h - IO_60HZ/50HZ_boldness_contrast ;initially(02h) ;bit2-7 NOT R/W
The firmware does initially set these registers to 02h, but may later change them to 00h or 02h (in response to bit0 of IO_AV_stat_detect_1).
For PAL pictures, bit1 affects boldness, and bit0 has no visible effect. For bugged PAL60 color errors, bit0-1 are somewhat changing the contrast of the color error.

FC96h/FCC2h - IO_60HZ/50HZ_ratio_whatever_A_lsb ;(61h,7Eh,5Bh/5Ch,84h,63h)
FC97h/FCC3h - IO_60HZ/50HZ_ratio_whatever_A_msb ;(08h,08h,03h/08h,08h,03h)
FCADh/FCD9h - IO_60HZ/50HZ_ratio_whatever_B_lsb ;(00h,00h,09h/00h,00h,06h)
FCAEh/FCDAh - IO_60HZ/50HZ_ratio_whatever_B_msb ;(00h,00h,00h/00h,00h,00h)
Unknown, these registers seem to have no visible effect. The firmware uses following fixed values, depending on screen ratio:

```
  NTSC at "16:9"    A=0861h, B=0000h
  NTSC at "4:3"     A=087Eh, B=0000h
  NTSC at (unused)  A=035Bh, B=0009h
  PAL  at "16:9"    A=085Ch, B=0000h
  PAL  at "4:3"     A=0884h, B=0000h
  PAL  at (unused)  A=0363h, B=0006h
```

Whereof, a "4:3" display defaults to using the "16:9" values (not the "4:3" values).

FC94h/FCC0h - IO_60HZ/50HZ_internal_unused1 (00h) ;
FC95h/FCC1h - IO_60HZ/50HZ_internal_unused2 (00h) ;
FCB1h/FCDDh - IO_60HZ/50HZ_internal_unused3 (14h) ;bit5-7:NOT RW
FCB2h/FCDEh - IO_60HZ/50HZ_internal_unused4 (00h) ;
FCB3h/FCDFh - IO_60HZ/50HZ_internal_unused5 (00h) ;
FCB4h/FCE0h - IO_60HZ/50HZ_internal_unused6 (00h) ;bit2-7:NOT RW
FCB5h/FCF1h - IO_60HZ/50HZ_internal_unused7 (00h) ;bit7:PAL60 minimal effect?
Not used by older firmware. Newer firmware does initialize them (but just using same values as they are having on power-up anyways). No visible effect (aside from FCB5h.bit7 slightly affecting PAL60 color errors... the error somewhat getting moved downwards a bit).

Screen Ratio Notes (4:3 or 16:9)...
The seven "ratio" registers exist for 60Hz and 50Hz each (ie. 14 registers in total), and their values depend on the selected screen ratio (4:3 or 16:9). Not quite clear if/how that's working:
My firmware has some tables with six different values for each register. Of those six values, the first two values seem to be associated with 16:9 and 4:3 (and the other four values are unused). Since having a 4:3 display, my firmware is using only one of the aforementioned two values (whereof, it is, oddly, using the 16:9 value (not the 4:3 value) for my 4:3 display).
Maybe that's implemented somewhat differently on 16:9 displays with actual screen ratio options in the user interface.
Also unknown, WHAT that ratio option is supposed to do: Maybe always stretching any picture to the full screen width, or maybe doing so only when receiving certain wide-screen movie signals, or whatever?

```
 _________________________ General Control Registers _________________________
```

FC90h - IO_VIDEO_control ;(should be set to 02h)

```
   bit0 On PAL50: causes VerticalScale (force NTSC resolution?),
   bit5 Swap AV colors Red & Blue
   bit7 MinorEffectOnRightScreenEdge
```

FC00h - IO_LCD_color_swap (should be set to 05h)
The firmware initializes this register to fixed value. Changing bit0-3 does somehow "swap" colors.
```
  0-3   Color swap for AV and OSD (ie. LCD databus mode)
  4-7   Unknown, no visible effect on PAL+PAL60 image
```

FCB7h - IO_VIDEO_something_1_lsb (should be set to 16h)
FCB8h - IO_VIDEO_something_1_msb (should be set to 01h)
FCBBh - IO_VIDEO_something_2_lsb (should be set to 1Ch)
FCBCh - IO_VIDEO_something_2_msb (should be set to 01h)
FCE3h - IO_VIDEO_something_3 (should be set to 01h) ;-bit1-2: AV v-position
FCE4h - IO_VIDEO_something_4 (should be set to 45h) ;-bit4: AV h-position
The firmware initializes these registers to fixed values. For whatever reason it's also ORing [FCE4h] by 40h (done so before switching between 16:9 and 4:3 screen ratio, despite of it containing 45h anyways).

FCEAh - IO_VIDEO_something_5 ;NOT R/W... but firmware writes to it!
Whatever. Looks unused/read-only (always reads FFh), but the firmware writes 00h to this register (within Timer 1 IRQ handler, and after switching between 16:9 and 4:3 screen ratio). Maybe the firmware write is doing some write-only acknowledge, or maybe it's just a bug.

```
 __________________________ More Internal Registers __________________________
```

FCE5h - whatever_status ;NOT R/W ;(always 36h ?)
FCE6h - whatever_status ;NOT R/W ;(always 02h ?)
Not used by firmware. The two bytes seems to be always 36h and 02h (with SNES PAL, with C64, and also without AV signal).

FCE7h - whatever ;bit2=R/W! ;other seven bits:NOT R/W (01h)
Not used by firmware. Contains mixed R and R/W bits.
```
  0     Unknown, read-only (1)
  1     Unknown, read-only (zero)
  2     Unknown, read/write-able (usually zero, no visible effect when changed)
  3-7   Unknown, read-only (zero)
```

FCE8h - whatever_status_msb ;NOT R/W
FCE9h - whatever_status_lsb ;NOT R/W
```
  with PSX/PAL:       toggles 11Dh/11Eh
  with Commodore C64  toggles 11Bh/11Ch
  with ZX Spectrum    toggles 11Ch/11Dh
  with SNES/PAL:      toggles 11Ch/11Dh
  with SNES/PAL60:    toggles 117h/118h (and has BlackWhiteDither/wrong color)
  without signal      increasing 0..1xxh
```
Not used by firmware.

FC01h - whatever ;no effect on PAL picture?
FC02h - whatever ;bit0,1,2:kills AV image, bit3-7: NOT R/W
Not used by firmware.

FC03h..FC10h - whatever, 0Eh bytes, not used by firmware, no visible effect
FC12h..FC46h - whatever, 35h bytes, not used by firmware, no visible effect

```
  FC09h.bit5-7:NOT RW  ;-
  FC0Dh.bit6-7:NOT RW  ;-
  FC0Fh.bit6-7:NOT RW  ;-
  FC13h.bit4-7:NOT RW  ;-
  FC19h.bit3-7:NOT RW  ;\
  FC1Bh.bit3-7:NOT RW  ;
  FC1Dh.bit3-7:NOT RW  ;
  FC1Fh.bit3-7:NOT RW  ;
  FC21h.bit3-7:NOT RW  ;
  FC23h.bit3-7:NOT RW  ;
  FC25h.bit3-7:NOT RW  ;
  FC27h.bit3-7:NOT RW  ;
  FC29h.bit3-7:NOT RW  ;
  FC2Bh.bit3-7:NOT RW  ;
  FC2Dh.bit3-7:NOT RW  ;
  FC2Fh.bit3-7:NOT RW  ;/
  FC31h.bit2-7:NOT RW  ;\
  FC33h.bit2-7:NOT RW  ;/
  FC44h.bit4-7:NOT RW  ;-
  FC46h.bit6-7:NOT RW  ;-
```

FC11h - Unused (open-bus) (1 byte)
FC47h..FC8Fh - Unused (open-bus) (49h bytes)
FCB9h..FCBAh - Unused (open-bus) (02h bytes)
FCEBh..FCFFh - Unused (open-bus) (15h bytes)
Probably unused I/O addresses.

AMT630A - FDxxh - Misc Registers (PWM,ADC,PLL,PIN,SPI-FLASH)

```
 ______________________________ PWM Registers ______________________________
```

PWM0 is wired to the backlight voltage generator. However, the original firmware fails to use the PWM feature; instead, it's simply switching the pin to always HIGH, resulting in a painfully bright image, and huge waste of energy. Using duty 50% (with total=1000h and high=0800h) does produce more moderate brightness, and does greatly reduce power consumption. Measured at 5V power line:

```
  PWM Duty               Whole System  --  Backlight Part
  duty 100% (aka HIGH) = 290mA (1.45W) --  234mA (1.17W)
  duty 50%              = 65mA  (0.33W) --  9mA   (0.05W)
  duty 0% (aka LOW)     = 56mA  (0.28W) --  0mA   (0W)
```

Note: The backlight doesn't work with too small total values (eg. backlight stays off at total=0100h and high=0080h).

FD1Fh - IO_PWM_enable_flags

```
  0     PWM0 Enable (0=Disable, 1=Enable) ;LCD Backlight
  1     PWM1 Enable (0=Disable, 1=Enable) ;LCD Config, SPI.DTA
  2     PWM2 Enable (0=Disable, 1=Enable) ;LCD Config, SPI.CLK and SPI./RESET
  3     PWM3 Enable (0=Disable, 1=Enable) ;LCD Config, SPI./CS
  4-7   PWM0..3 Invert Flags (0=Normal/Duty=HIGH, 1=Invert/Duty=LOW)
```

Aside from the enable bits, one will also need to switch the pins to PWM mode (via IO_PIN_xxx), and also configure the MODE_A/B bits in SFR registers.

FD20h/FD22h/FD24h/FD26h - IO_PWM#_duty_total_lsb (#=0..3)
FD21h/FD23h/FD25h/FD27h - IO_PWM#_duty_total_msb (#=0..3)

```
  0-15  Total Duty, in 26MHz cycles (for HIGH and LOW periods)
```

FD28h/FD2Ah/FD2Ch/FD2Eh - IO_PWM#_duty_high_lsb (#=0..3)
FD29h/FD2Bh/FD2Dh/FD2Fh - IO_PWM#_duty_high_msb (#=0..3)

```
  0-15  Duty HIGH, in 26MHz cycles (for HIGH period) (or LOW when inverted)
```

```
 ________________________________ ADC Registers ________________________________
```

ADC Channel 0 is used for Keyboard input, the keypads consist of several buttons with different resistors (allowing to access multiple buttons via a 2-wire cable). The most common keypad seems to be the "3KEY" keypad with three buttons, using these ADC values:

```
  04F0h..05BFh  (avg=0564h)   ;1.0K ohm (button +)     (used as Up/Right)
  07A0h..086Fh  (avg=0804h)   ;2.0K ohm (button menu)  (Menu)
  0AD0h..0B9Fh  (avg=0B38h)   ;4.7K ohm (button -)     (used as Down/Left)
```

Further possible values (not supported by original firmware) would be:

```
  0300h..03D7h  (avg=03A5h)   ;Menu+Plus+Minus
  03D8h..046Fh  (avg=040Bh)   ;Menu+Plus
  0450h..04EFh  (avg=04B4h)   ;Plus+Minus
  0650h..06FFh  (avg=06A1h)   ;Menu+Minus
  0FA0h..0FFFh  (avg=0FF5h)   ;(when no button pressed)
  xxxxh                       ;(invalid, might happen on transitions)
```

CAUTION: Above is for a 3.5" display. Whilst, for 4.3" displays, Plus and Minus are swapped (ie. the same 3KEY board seems to be mounted the other way around, probably due to different case design for 4.3" screens).
Sensing multiple simultaneously pressed buttons is possible, though not too recommended (first of, it's difficult to press more than one of the tiny buttons, and, some values are a bit close together which might cause misreadings in case values differ due to resistor tolerance or temperature).
Unlike digital signals (which are always 0 or 1), analog signals might transition through nonsense values when pushing/releasing buttons; to avoid such problems read more than one ADC value, and then check that at least two continous readings are reflecting the same button, or, check that at least half of the last some readings are reflecting the same button(s), the latter method should also work for multiple buttons, including cases like one button being held down while another button is changing.
Note: The firmware contains definitions for 23 different keypads (some having only 2 buttons, others having up to 7 buttons, including "hotkeys" for volume

up/down, manual power/standby, etc., and some using buttons with ADC value 0FFFh, so those are apparently wired with VCC/GND swapped).
Further alternate inputs would be IR remote controls (though unknown how to handle to wire that to the board, and unknown how to use RC-5 protocol; currently only the messy NEC protocol is known), and, another "standard" input (unintended by the manufacturer) would be a serial mouse on the UART port.
Other than that, one could brew-up any kind of nonstandard stuff for ps/2 keyboards, gamepads, digital joysticks, analog joysticks, or whatever. Some display include unused touch screen membranes (or they did so, but were removed with scissors, as seen on bigby's photo), but connecting the four touch screen signals to the two spare ADC inputs would require some analog multiplexing.

FDB0h - IO_ADC_ctrl_lsb
FDB1h - IO_ADC_ctrl_msb

```
  0      Unknown (initially set to 0, this bit isn't R/W though) (R) or (W)?
  1-2    Unknown (initially set to 0, hangs ADC when setting bit1 or bit2)
  3      Channel 0 Run  ;\firmware sets ONE of these bits on conversion start
  4      Channel 1 Run  ; (and clears the other three bits)
  5      Channel 2 Run  ; (required to be "1", ie. start/enable?)
  6      Unknown  VCOM? ;/
  7      Unknown (initially set to 0)
  8-11   Unknown (initially set to 1111b) (not required, can be 0)
  12     Unknown  VCOM? ;\firmware sets ONE of these bits on conversion start
  13     Channel 0      ; (and clears the other three bits)
  14     Channel 1      ; (not required to be "1", can be "0")
  15     Channel 2      ;/
```

Firmware does rewrite bit3-6 and bit12-15 upon each conversion start, however, actually that's required only once (and hardware will then repeatedly throw new ADC conversion results).
Bit8-11 and Bit12-15 might select some per channel mode, maybe channel priority, maybe reference voltage, or maybe speed select (in case FDC8h and FDCAh are alternate speed's)?
ADC hangs if bit8-11 AND bit12-15 are BOTH zero (but works if either one (or both) of them is nonzero).

FDB8h - IO_ADC_status_lsb (Read-only, except: Write 0 to ack)
FDB9h - IO_ADC_status_msb (Read-only, except: Write 0 to ack)

```
  0-1   Unknown, maybe channel 0 related
  2     Channel 0 Ready                (0=Busy, 1=Ready)
  3-4   Unknown, maybe channel 1 related
  5     Channel 1 Ready                (0=Busy, 1=Ready)
  6-7   Unknown, maybe channel 2 related
  8     Channel 2 Ready                (0=Busy, 1=Ready)
  9-10  Unknown, maybe channel VCOM? related
  11    Channel VCOM? Ready            (0=Busy, 1=Ready)
  12-15 Unknown
```

FDB6h - IO_ADC_config_4 - should be set to FFh ;<-- as so disables ADC IRQs?
FDB7h - IO_ADC_config_5 - should be set to FFh
Firmware sets these to FFh. Maybe IRQ Disable flags corresponding to bits in IO_ADC_status registers, ie. bit0-2 for channel 0, and so on?

FDBCh/FDBEh/FDC0h - IO_ADC_input_#_lsb (#=0,1,2) (R)
FDBDh/FDBFh/FDC1h - IO_ADC_input_#_msb (#=0,1,2) (R)

```
  0-11   Conversion result (0..FFFh)
  12-15  Always 0 (assuming ADC cannot be configured to more than 12bit)
```

FDB2h - IO_ADC_config_1 - should be set to 20h ;DANGER:bit1 ;bit6-7:NOT R/W
FDB4h - IO_ADC_config_2 - should be set to 22h ;DANGER:bit7
FDB5h - IO_ADC_config_3 - should be set to 37h ;DANGER:bit0-1
FDC8h - IO_ADC_config_6 - should be set to FFh ;\maybe another speed?
FDC9h - IO_ADC_config_7 - should be set to 0Fh ;/maybe for VCOM?
FDCAh - IO_ADC_config_8 - should be set to 00h
The firmware initializes these to fixed values. Changing some of those bits can hang the ADC (see DANGER bits).

FDCCh - IO_ADC_speed_lsb - should be set to FFh
FDCDh - IO_ADC_speed_msb - should be set to 0Fh ;upper bits make ADC slower
ADC Conversion speed, firmware sets this to 0FFFh (4095), and alongsides sets IO_PLL_adc_clk_divider to 18h (24). With that values, the conversion rate is around 137Hz (apparently computed as 27MHz/4095/24/2).

FDB3h - whatever ADC related status, 1 byte (R)
FDBAh..FDBBh - whatever ADC related status, 2 bytes (R) maybe VCOM result?
FDC2h..FDC7h - whatever ADC related status, 6 bytes (R)
FDCBh - whatever ADC related status, 1 byte (R)
Whatever, not used by firmware, not read/write-able. Reading these registers always seems to return 00h.

FDCEh - whatever ADC related control/status (not used by firmware)

```
  0-6   Unknown, read-only (contains value 1Fh)         (R)
  7     Unknown, read/write-able                        (R/W)
```

FDCFh - whatever ADC related control (not unused by firmware)

```
  0-3   Unknown                                         (R/W)
  4-7   Unused                                          (not R/W)
```

```
 _____________________________ SPI FLASH Registers _____________________________
```

FDD0h - IO_SPI_transfer_mode
Set ONE bit (used: 04h,08h,10h,40h,80h) ;DANGER

```
  01h = Manual read JEDEC Chip ID
  02h = Manual read Write-Protect-Status
  04h = Manual write Write-Protect-Status
  08h = Manual send FLASH command (used for WREN and ERASE)
  10h = Manual read FLASH-to-CPU
  20h = unknown (not used by firmware)
  40h = DMA read FLASH-to-FONT
  80h = DMA write XRAM-to-FLASH (max 100h bytes at once)
```

Note: For FLASH-to-FONT, one should disable font rendering via SFR_IO_memory_system during DMA (the OSD will then draw all windows as if FONT memory were zerofilled, eg. using the BGMAP's BG attr color for 1bpp windows).
Note: FLASH must be erased (in 1000h-byte units) before writing, writing from XRAM is restriced to 100h-byte pages (and also to 800h-byte XRAM size), but, one can erase 1000h bytes, and then write sixteen 100h-byte snippets.

FDD1h - IO_SPI_wprot_stat_write
Allows to change the nonvolatile write-protect bits in the FLASH chip's status register. The firmware is taking excessive use of this register: Instead of leaving the firmware block permanently protected (via value 24h), it's changing the write protect status before & after each write (meaning that the status bits will get worn out long before the actual data sectors get worn out, and also meaning that accidental jumps to the write function will automatically unlock writing).

```
  0    BUSY   Busy (should be 0)
  1    WEL    Write Enable Latch (should be 0)
  2-4  BP0-2  Number of 64Kbyte Blocks to protect (0=none, 1..7=see below)
  5    TB     Protect Top/Bottom Blocks (0=Top, 1=Bottom)
  6    -      Reserved (should be 0)
  7    SRP    Status Register Protect when /WP=LOW (0=No, 1=Protect if /WP=LOW)
```

The BP bits allow to specify the amount of write-protected blocks (either at top or bottom addresses, depending on the TB bit) (if the amount of blocks is same or bigger than the chipsize, the ALL blocks are protected, and TB bit is don't care).

```
  BP=00h      Protect nothing (unlock all blocks)
  BP=01h      Protect 1 block (64Kbytes)
  BP=02h      Protect 2 blocks (128Kbytes)
  BP=03h      Protect 4 blocks (256Kbytes)
  BP=04h..07h Same as 00h..03h                 ;-on W25D10/W25D20
  BP=04h      Protect 8 blocks (512Kbytes)      ;\on W25D40/W25D80
  BP=05h..07h Protect all blocks (1024Kbytes)   ;/
```

The older firmware uses a hardcoded protection value of 0Ch (locking ALL blocks on W25D10/W25D20, but accidentally locking only the UPPER some blocks on W25D40, whereas, my board WAS shipped with a 25D40 chip).
The newer firmware tries to leave the firmware block always protected (unless the settings are stored in the same block, which is actually happening due to a bug in the firmware's chip detection).
NOTE: The MK 25D40 identifies itself with same chip ID as W25D40... but it doesn't seem to allow to set the TB flag in bit5, unknown if/how protection works on that chip (except, protecting ALL blocks seems to work).

FDD2h - IO_SPI_manual_data_read (R)
Data, used for manual data read (and also for write-protect-status read). Not used for DMA access.

FDD3h - IO_SPI_chip_id_read_msb (R) (initially 00h)
FDD4h - IO_SPI_chip_id_read_mid (R) (initially 00h)
FDD5h - IO_SPI_chip_id_read_lsb (R) (initially 00h)

```
  0-6     Maker bit1-7    (EFh/2 for Winbond)
  7-14    Device Capacity (13h for Winbond W25D40)
  15-22   Device Type     (40h for Winbond W25D40)
  23      Maker bit0 (?)  (EFh and 01h for Winbond)
```

As shown above, the register is weirdly right-rotated, to fix that hardware glitch: left-rotate the whole 24bit value (unknown if the carry-in really gives intact maker.bit0 (JEDEC maker codes are actually 7bit+parity, so maybe the weird rotation was done intentionally for stripping the parity bit).
Used by newer firmware only (the firmware does totally discard bit23, ie. it's merley doing "mul2" instead of "rol").

FDD6h - IO_SPI_flash_addr_lsb
FDD7h - IO_SPI_flash_addr_mid
FDD8h - IO_SPI_flash_addr_msb

```
  0-23  Address in SPI FLASH memory (for read/write/erase operations)
```

FDD9h - IO_SPI_dma_ram_addr_lsb
FDDAh - IO_SPI_dma_ram_addr_msb

```
  0-14  Address in XRAM or FONT memory (for DMA, not used for manual CPU read)
  15    Unknown, used! (maybe DIRECTION, or maybe XRAM vs FONT select?)
```

FDDBh - IO_SPI_dma_length_lsb
FDDCh - IO_SPI_dma_length_msb

```
  0-15  Transfer length in bytes (MINUS 1) (for DMA, not used for manual read)
```

FDDDh - IO_SPI_ready_flags (write 00h to reset)

```
  0-7   Ready flags (corresponding to start.bits in FDD0h) (1=Ready)
```

Maybe these bits (or FDDFh) do trigger IRQ 0013h (aka ext.1 interrupt)?

FDDEh - IO_SPI_kick_stop_reset
Write 80h to start/stop/reset? DANGER: setting bit6 hangs CPU (once caused weird OSD roll).

FDDFh - IO_SPI_status_ready (R)

```
  0-6   Unknown/unused
  7     FLASH Status (0=busy, 1=ready)
```

FDE4h - IO_SPI_command_write (D8h)
Used for manually writing certain commands (in most/other cases, the memory system is automatically sending appropriate commands). Used values are:

```
  04h  Write Disable (WEL/WREN=0) (used for whatever reason)
  06h  Write Enable (WEL/WREN=1) (used before Erase, Write,and WriteWprot)
  20h  Erase 1000h-byte Sector (used when saving settings)
  D8h  Erase 10000h-byte Block (used in an unused table entry in old firmware)
  D7h  Erase whatever?         (used in an unused table entry in old firmware)
```

FDE0h - whatever SPI control, DANGER! crashes CPU (not used by firmware) 03h

```
  0     hangs and draws TWO garbage characters on OSD?
  1     hangs and draws TWO garbage characters on OSD?
  2     hangs and draws TWO garbage characters on OSD?
  3     jumps to a pushed retadr at some/several stages higher stack level?
  4     hangs and draws ONE garbage character on OSD?
  5     hangs
  6     hangs
  7     hangs
```

FDE1h..FDE3h - whatever SPI control, 3 bytes (02h,05h,01h)
FDE5h..FDE7h - whatever SPI control, 3 bytes (9Fh,06h,07h)
Whatever, not used by firmware. Might be 24bit FLASH address(es), but seem to have no effect the XRAM memory window?
Or rather... looks as if FDE0h..FDE7h are SPI commands for Read, Write, GetID, etc. (which could be changed for SPI chips with different command set).

FDF0h - Unknown (not used by firmware) (07h)
Unknown, not used by firmware, seems to have no effect when changed.

FDF1h - IO_SPI_upper_32k_code_base (not used by firmware)

```
  0-3  SPI Base Address for CODE at 8000h-FFFFh in 32Kbyte units (usually 01h)
  4-7  Probably MSBs of above (no effect on 512Kbyte flash chip)
```

Note: There's another/unrelated SPI bus for LCD display configuration (which is done via manually generating SPI clk/dta/select/reset signals on some other pins).

```
  _______________________________ PIN Registers _______________________________
```

Most of the registers below are probably 4bit mode value for each chip pin...

FD30h - IO_PIN_maybe (not used by firmware) (maybe P04_P05 = UART/I2C?)
FD31h - IO_PIN_P06_P07_pwm ;PWM2/PWM3
FD32h - IO_PIN_P10_P11_spi_flash ;DANGER (11h)
FD33h - IO_PIN_P12_P13_spi_flash ;DANGER (11h)
FD34h - IO_PIN_P14_P15_lcd ;RGB
FD35h - IO_PIN_P16_P17_lcd ;RGB
FD36h - IO_PIN_P20_P21_lcd ;RGB
FD37h - IO_PIN_P22_GPIO0_lcd ;RGB ;bit0:addDarkBLUE, bit4:addMoreBLUE
FD38h - IO_PIN_GPIO1_GPIO2_lcd ;RGB
FD39h - IO_PIN_GPIO3_GPIO4_lcd ;RGB
FD3Ah - IO_PIN_P23_P24_lcd ;RGB ;bit4:addDarkGREEN
FD3Bh - IO_PIN_P25_P26_lcd ;RGB ;bit0:addMoreGREEN, bit4:addManyGREEN
FD3Ch - IO_PIN_GPIO5_GPIO6_lcd ;RGB
FD3Dh - IO_PIN_GPIO7_GPIO8_lcd ;RGB
FD3Eh - IO_PIN_GPIO9_GPIO10_lcd ;RGB
FD3Fh - IO_PIN_P27_P30_lcd ;RGB ;bit0:addDarkRED, bit4:addMoreRED
FD40h - IO_PIN_P31_P32_lcd ;DCK/DOE? ;bit0:StopDotClk, bit1:HIRES, bit2:LORES

FD41h - IO_PIN_P33_P34_lcd ;HOS/VOS? ;bit0-5:Kill/Move/Shake/Freeze Image
FD42h - IO_PIN_P35_P36_pwm ;PWM0/PWM1 ;bit0-2:BacklightBlack
FD43h - IO_PIN_P37_xxx_pwm ;PWM2'/xxx
FD44h - IO_PIN_maybe (set to 01h by firmware) ;bit0:force dotclk stuck/low
FD45h - IO_PIN_maybe (set to zero by firmware)
FD46h - IO_PIN_maybe (set to zero by firmware)
FD47h - IO_PIN_maybe (set to zero by firmware)
FD48h - IO_PIN_maybe (set to zero by firmware)
FD49h - IO_PIN_maybe (set to zero by firmware)
FD4Ah - IO_PIN_maybe (set to zero by firmware)
FD4Bh - IO_PIN_maybe (set to zero by firmware)
The firmware contains functions for switching the PWM pins to PWM mode or to manual output High/Low mode.
The firmware initializes all SPI and LCD registers to 11h. The pins for SPI and LCD are following the chip's physical pin-ordering, ie. with GPIOxx pins inserted within the Pxx ports (confirmed by messing with the MSBs of the RGB signals).
Some of the other/unknown registers might be related to Infrared REMOTE pin, the three ADC input pins, and to the two UART/I2C pins, and possibly some further LCD/voltage related pins.

FD50h - IO_whatever_FD50h ;(initially 00h, then set to 0Bh)
```
  bit5:KillTftUpdating (force dotclk stuck/low)
```

FD51h..FD5Ah - IO_whatever_zerofilled, 10 bytes ;fixed (zerofilled)
```
  FD59h.bit2 add some zigzag noise on dotclk ;\normally square wave gets
  FD59h.bit3 add more zigzag noise on dotclk ;/zigzag at raising/falling edges
```
Maybe 2bit signal strengths for all PINs?

FD4Ch..FD4Fh - whatever, 4 bytes
FD5Bh..FD5Fh - whatever, 5 bytes
Whatever, read/write-able, not used by firmware.

LCD Databus type is controlled by following registers:
```
  FC00h        - IO_LCD_color_swap   (color swap, eg. RGB vs BGR or so)
  FD34h..FD43h - IO_PIN_xxx_xxx_lcd  (maybe HV vs DEN mode, or RGB vs YCbCr)
  FD50h        - IO_whatever_FD50h   (no visible effect...?)
```
Known settings are:
```
  Tianma   24bit RGB+HV  : FC00h=05h, FD34h..FD43h=11h, FD50h=0Bh
  Innolux  24bit RGB+DEN : FC00h=00h, FD34h..FD43h=22h, FD50h=0Fh
```

```
 ________________________________ PLL Registers ________________________________
```

The "PLL" registers are configuring several clocks (for AV, LCD, CPU, PWM, ADC, etc.), allowing to multiply & divide the 27MHz crystal clock, and to enable/disable some clocks. The firmware initializes several registers during initialization (in some cases twice writing the same value, which is probably not

required, and in some cases writing 2-3 different values during init, which might be a bug, or it's required for proper wake-up).
Apart from initialization, the firmware does also alter FD11h/FD12h/FD13h when entering/leaving standby mode, probably for some sort of power-saving, either disabling or slowing down certain clocks.

FD11h - IO_PLL_11h_used (init: MOV FFh, on: OR E0h, off: AND 1Fh) ;Sysclk?

```
0?   hangs CPU?
1-2? OSDcharerror+hang?
3    unknown, no visible effect
4    OSD_BG_ONLY (no TEXT)
5    scanlinefreeze(AV+OSD)
6-7  unknown, no visible effect
```

FD12h - IO_PLL_12h_used (init: MOV FFh, on: OR EFh, off: AND 38h) ;ADC

Seems to contain enable flags for AV, TFT, ADC, PWM clocks.

```
0-1  unknown, no visible effect
2    AV filled by most-recent AV pixel color (OSD+backdrop still work)
3    kills TFT screen output (dotclk switched HIGH permanently)
4    DANGER: hangs ADC
5    stops PWM (randomly gets stuck at MAX or MIN backlight level)
6-7  unknown, no visible effect
```

FD13h - IO_PLL_13h_used (init: MOV FFh, on: OR FFh, off: AND 00h) ;Flag?

There seem to be no noticable effects when changing this register, though it might affect some less obvious timings? Apart from writing to FD13h, the firmware is also reading FD13h in several places (for checking if FD11h/FD12h/FD13h are currently in standby mode).

FD19h - IO_PLL_19h_used ;(initially ORed by 81h, later set to 83h)

The firmware initializes this register twice during initialization: First to power-up value ORed by 81h, then setting the register to 83h.

```
bit0 NoSignal, withSmallVsyncRoll
bit6 Darker(AV+OSD) -- PWM is (almost) twice as fast as usually
bit7 NoSignal, occassionally/shortly AV scanlines shown twice/at half width
```

FD0Bh - IO_PLL_0Bh_used ;(twice set to 40h)

The firmware initializes this register to fixed value, messing with it can have several effects.

```
bit0     force NTSC color, or somewhat other/wrong color clock?
bit1,3-7 force NoSignal
bit2     force gray AV at wrong Xloc
bit0,1   C64: temporarily enables C64 image
```

FD0Dh - IO_PLL_0Dh_cfg ;fixed (F0h) ;15kHz, MSBs: snow+rolling sync?

```
bit4: flipped changes dotclk to 16.12MHz (instead 8.06)  ;\shift/divider
bit5: flipped changes dotclk to 32.26MHz (instead 8.06)  ;/for dotclk?
bit6-7: no signal (no AV image)
bit7: no effect on dotclk... but much more white snow
```

FD0Eh - IO_PLL_0Eh_used ;(set to 20h or 2Ch) ;scanline freeze?

The firmware initializes this register thrice during initialization: First to 20h, then to 20h again, and finally to 2Ch.

```
  bit2:  freeze dotclk (get stuck in LOW or HIGH state)
  bit3:  freeze snow (backdrop get stuck with all WHITE or BLUE/BLACK pixels)
```

FD0Ah - IO_PLL_dotclk_multiplier ;fixed (2Bh) ;15kHz (also SNOW-boldness?)

```
 FD0Ah dotclk:  MULTIPLY by N
  2Bh = (default)      8.06MHz    (0.187MHz * 43)
  2Ah = (bit0 flipped) 7.88
  29h = (bit1 flipped) 7.68
  2Fh = (bit2 flipped) 8.82
  23h = (bit3 flipped) 6.56       (0.187MHz * 35)
  3Bh = (bit4 flipped) 11.06      (0.187MHz * 59)
  0Bh = (bit5 flipped) 2.06       (0.187MHz * 11)
  6Bh = (bit6 flipped) 15.74      MAX (0.187MHz * 84)
  ABh = (bit7 flipped) 15.74      MAX (0.187MHz * 84)
```

FD0Fh - IO_PLL_dotclk_divider_1 ;fixed (03h) ;15kHz ...inserts VSYNC's?

```
 FD0Fh dotclk:  DIVIDE by N (with some odd effects in some cases)
  03h = (default)      8.06MHz    (15.72MHz / 1.95)   (24.20MHz / 3)
  02h = (bit0 flipped) 12.10MHz   (15.72MHz / 1.3)    (24.20MHz / 2)
  01h = (bit1 flipped) 15.72MHz   (15.72MHz / 1)      (24.20MHz / 1.54) ;odd
  07h = (bit2 flipped) 3.46MHz    (15.72MHz / 4.54)   (24.20MHz / 7)
  0Bh = (bit3 flipped) 2.20MHz    (15.72MHz / 7.14)   (24.20MHz / 11)
  13h = (bit4 flipped) 1.27MHz    (15.72MHz / 12.37)
  23h = (bit5 flipped) 0.69MHz    (15.72MHz / 22.78)  (24.20MHz / 35)
  43h = (bit6 flipped) unstable (0.3-0.4MHz)          (stutters)     ;<-- odd
  83h = (bit7 flipped) 15.74MHz   MAX                 (MAX)          ;<-- odd
```

FD14h - IO_PLL_dotclk_divider_2 ;fixed (03h) ;similar effects as FD0Fh

```
 FD14h dotclk:  DIVIDE by N
  03h = (default)      8.06MHz  (24.20 / 3)
  02h = (bit0 flipped) 12.10MHz (24.20 / 2)
  01h = (bit1 flipped) 24.20MHz (24.20 / 1) ;<-- unlike FD0Fh
  07h = (bit2 flipped) 3.46MHz  (24.20 / 7)
  0Bh = (bit3 flipped) 2.20MHz
  13h = (bit4 flipped) 1.27MHz  (24.20 / 19)
  23h = (bit5 flipped) 0.69MHz
  43h = (bit6 flipped) 0.36MHz             ;<-- unlike FD0Fh
  83h = (bit7 flipped) 24.20MHz            ;<-- unlike FD0Fh
```

FD15h - IO_PLL_dotclk_divider_3 ;fixed (02h) ;similar effects as FD0Fh

```
 FD15h dotclk:  DIVIDE by N
  02h = (default)      8.06MHz   (16.12 / 2)
  03h = (bit0 flipped) 5.38MHz   (16.12 / 3)
  00h = (bit1 flipped) 16.12MHz
  06h = (bit2 flipped) 2.68MHz   (16.12 / 6)
  0Ah = (bit3 flipped) 1.61MHz   (16.12 / 10)
  12h = (bit4 flipped) 0.89MHz
  22h = (bit5 flipped) 0.47MHz
  42h = (bit6 flipped) 0.24MHz
  82h = (bit7 flipped) 16.12MHz
```

FD10h - IO_PLL_10h_cfg ;fixed (twice set to 04h) ;if changed: NoSignal
FD16h - IO_PLL_16h_cfg ;fixed (03h) ;unknown (no visible effect)
FD17h - IO_PLL_adc_clk_divider ;fixed (18h) ;bit5-6:SlowerADC, bit7=HangsADC
FD1Ah - IO_PLL_1Ah_cfg ;fixed (08h) ;unknown (no visible effect)
The firmware initializes these registers to fixed values.

FD0Ah,FD0Fh,FD16h seem to be related to vertical resolution/scaling, used values are 2Bh,03h,03h for 320x240, and A8h,09h,0Ah for 480x272, the three bytes are apparently one multiplier and two dividers or vice-versa (confirmed by replacing 2Bh,03h,03h by stuff like 2Bh*3,03h*3,03h*3, which didn't affect the picture).

FD00h - IO_PLL_00h (00h)
FD01h - IO_PLL_01h_cpu (01h) ;bit0:hangs CPU
FD02h - IO_PLL_02h_zoom (00h) ;bit0:zoom/wobble
FD03h - IO_PLL_03h (00h)
FD04h - IO_PLL_04h (00h)
FD05h - IO_PLL_05h (00h)
FD06h - IO_PLL_06h (00h)
FD07h - IO_PLL_07h (00h)
FD08h - IO_PLL_08h ;NOT R/W (value 00h) (R)
FD09h - IO_PLL_09h ;NOT R/W (value 00h) (R)
FD0Ch - IO_PLL_0Ch (33h)
FD18h - IO_PLL_18h_pwm (FFh) ;bit7:ForceMaxBacklight/PWMdimmingOFF
FD1Bh - IO_PLL_1Bh (FFh)
FD1Ch - IO_PLL_1Ch (FFh)
FD1Dh - IO_PLL_1Dh (00h)
FD1Eh - IO_PLL_1Eh (00h)
The firmware doesn't use these registers.

Unused Registers

FD60h..FDAFh - Unused (open-bus) (50h bytes)
FDE8h..FDEFh - Unused (open-bus) (08h bytes)

FDF2h..FDFFh - Unused (open-bus) (0Eh bytes)
Probably unused I/O addresses.

AMT630A - FExxh - AV Registers (Composite Video Input)

```
_____________________________ AV Status Registers _____________________________
```

FE26h - IO_AV_stat_detect_0 (R)

```
  video DETECT... OFTEN used (bit1,bit2 tested)
```

FE27h - IO_AV_stat_detect_1 (R)

```
  ...status (boldness, bit0 tested, used by only ONE opcode)
```

FED0h - IO_AV_stat_detect_2 (R)

```
  ...status (sharpness, bit2-5)
```

FE28h - IO_AV_stat_framerate_flag (R)

```
  bit2 = PAL/NTSC and/or 50/60 Hz
```

FE2Ah - IO_AV_stat_signal_detect (R)

```
  ...status (bit6 and bit0-3 used)
  bit0,1,2,3=ErrorFlag(s))
  bit4:Have Signal on currently selected pin    ;\CVBS1 and CVBS3 inputs
  bit6:Have Signal on currently de-selected pin ;/
```

FEAAh - IO_AV_stat_sensitivity_msb (R)
FEABh - IO_AV_stat_sensitivity_lsb (R)

```
  16bit status/counter?  (NoSignal:0FFFh, C64:0287h..028Bh=647..651)
```

```
__________________________ AV Misc Control Registers __________________________
```

FE01h - IO_AV_ctrl_whatever_1
bit1 set/cleared, bit4 set by firmware.

```
  bit0   ForcePALcolors_ButYetPAL60getsWrongYloc,
  bit2   ForcePALcolors_But_Only_If_FE00h.bit7 is toggled
  (similar as FE01h.bit0, but HERE bit2 has NO effect if FE00h.bit7=x)
```

FE54h - IO_AV_ctrl_whatever_2

```
  initially 00h, bit6 is manipulated later on
```

FEA0h - IO_AV_force_resync

bit0 manually pulsed with SLOW DELAYS by firmware.

```
  bit0   sometimes causes NoSignal, and simetimes FreezesCurrentScanline
         (THAT affecting BOTH OSD AND AV !!!)
  bit1   move AV image 1pix UP, and ca.8pix RIGHT
  bit2-7 NOT R/W
```

C64: UNTOGGLING bit7 does SHOW C64 image for short moment (uh, but b¡t7 isn't R/W, how did I do that?).

FECBh - IO_AV_ctrl_artifacts

Firmware sets this to 00h, 02h, or 06h.

```
  bit0: NICE: LessPalArtifactsOnSNES
  bit1: Similar as bit0, plus magenta-line at bottom of red-snes-screen
```

```
 ______________________ AV Sensitivity Control Registers ______________________
```

FE15h - IO_AV_ctrl_sensitivity_0

```
  set to 00h=max, 05h=med, 09h=low, also checked if 05h
  bit2,3:PixelBoldness
```

FED5h - IO_AV_ctrl_sensitivity_1

```
  initially B1h, modified/used later
```

```
 ____________________ AV Input Select and On/Off Registers ____________________
```

Used to select the AV input. The chip does support three AV inputs (AV1,AV2,AV3), but most boards have the AV2 input unused (GNDed, maybe intended as shielding between the other two pins), and renamed AV3 to AV2 in the GUI. The overall input selection flowchart is:

```
  Clear IO_AV_video_on_off bit3,4
  Change IO_AV_input_select_reg_0 bit6,7
  Set IO_AV_video_on_off bit3,4
  Change IO_AV_input_select_reg_1 bit4,5
```

Used values are shown below (looks as if newer firmware has swapped AV1+AV3 (though without swapping the corresponding bits in IO_AV_stat_signal_detect, unknown if/how that's working) and firmware also changed the dummy bits for unused AV2 input somehow).

```
  - - -    IO_AV_video_on_off
  - - -    |              IO_AV_input_select_reg_0
  - - -    |              |              IO_AV_input_select_reg_1
  Input    FED7h.bit3-4  FED8h.bit6-7  FEDCh.bit4-5
  OLDER FIRMWARE:
  AV1      0-then-3      2             0   ;<-- CVBS1
  AV2      0-then-3      2             3   ;<-- stripes when CVBS3 has signal
  AV3      0-then-3      0             2   ;<-- CVBS3
  Invalid  0-then-3      3             3   ;<-- stripes when CVBS3 has signal
  NEWER FIRMWARE:
  AV1      0-then-3      0 !           2 ! ;<--
  AV2      0-then-3      0 !           3   ;<--
  AV3      0-then-3      2 !           0 ! ;<--
  Invalid  0-then-3      3             3   ;<--
```

Unknown if there's some way to configure a pin-pair as S-Video input (older AMT630 chips did support that, but AMT630A specs don't mention any S-Video support).

FED7h - IO_AV_video_on_off

lcd-control/enable or so

```
  bit0+1+6+7:disable SNOW (freeze backdrop-color or snow-color,
             and occassionally also freeze current OSD scanline!)
```

FED8h - IO_AV_input_select_reg_0

```
  C64: bit7=whitish stripes when C64 ON
  ;SNES: bit7=ShortlyWhiteStripes...ThenAllWhite (looks as if snow/random
             generator ends up with all-white pixels)
```

FEDCh - IO_AV_input_select_reg_1

```
  bit4: Toggle=BlueBackdrop(without snow),
        Untoggle: ShortlyNoSignal(with snow)ThenNormalPicture
  bit5: NoSignal, or, if C64 powered on AV2: white picture with
                      blue/rolling hsync/vsync signals?
  bit6: BlueBackdrop(without snow, except a VERY FEW random/snow pixels
                     in some rolling-screen-half)
  bit6: Like bit6, but with a little bit more random/now pixels)
  bit7: C64: minimal SNOW when C64 ON
```

```
 ____________________________ AV Config Registers ____________________________
```

FE04h - IO_AV_config_FE04h - (should be set to 30h)
FE05h - IO_AV_config_FE05h - (should be set to 40h)
FE13h - IO_AV_config_FE13h - (should be set to 1Eh)
FE56h - IO_AV_config_FE56h - (should be set to 00h)
FE83h - IO_AV_config_FE83h - (should be set to FEh)
FEB5h - IO_AV_config_FEB5h - (should be set to 67h)
FEBAh - IO_AV_config_FEBAh - (should be set to FFh)
FED9h - IO_AV_config_FED9h - (should be set to bit4-5 cleared, bit6 set)
FEDBh - IO_AV_config_FEDBh - (should be set to bit7 cleared)

The firmware initializes these to fixed values. Changing the bits seems to have no visible effect.

FEC9h - IO_AV_config_FEC9h - (should be set to 01h or left unchanged)

Older firmware leaves this register unused, newer firmware sets it to 01h (although it's 01h on power-up anyways). Changing bits has some visible effects:

```
  bit1   Less Boldness
  bit6   Dithered All Green Image (GREEN !!!)
```

```
 _______________________ AV Internal Status Registers _______________________
```

FE20h..FE21h - 2 bytes - Status (R)
FE84h..FE89h - 6 bytes - Status (R) (two negative 16bit values?, and sth)
FE90h..FE9Fh - 16 bytes - Status (R)
FEA3h..FEA9h - 7 bytes - Status (R)
FEACh..FEB1h - 6 bytes - Status (R)
FEC5h..FEC7h - 3 bytes - Status (R) (values 00h)
FECEh..FECFh - 2 bytes - Status (R) (reads as DFh,73h with PAL/SNES)
FEDDh..FEEBh - 15 bytes - Status (R)
FEF0h..FEFDh - 14 bytes - Status (R) (not FFh)
FE7Eh - 1 byte - Status (R) (value FFh)
FE8Ch - 1 byte - Status (R) (value 17h)
FEA1h - 1 byte - Status (R) (value Fxh)
FEB3h - 1 byte - Status (R)
These registers contain some status information. The firmware doesn't use them.
Note: FEEAh..FEEBh are usually FFh, but FEEBh can become EFh when (un-)plugging AV cable.

```
 ______________________ AV Internal Control Registers ______________________
```

FE42h - IO_AV_secret_control (default=20h, better=00h)

```
  bit0-2 Slight horiz.position changes
  bit3   BlurRightScreenEdge
  bit4   NoSignal
  bit5   C64 does PERFECTLY SHOW C64 IMAGE! (still shows SNOW if C64=Off)
  bit6   Wrong Color (Lightblue instead Red in SnesDiag, Washy if C64 on
         AV2... hmmm or is that S-Video with Chroma from AV2?)
  bit7   no visible effect
```

FE00h - 1 byte
FE02h - 1 byte
FE03h - 1 byte
FE06h..FE12h - 13 bytes
FE14h - 1 byte
FE16h..FE1Fh - 10 bytes
FE22h..FE25h - 4 bytes
FE29h - 1 byte
FE2Bh..FE2Fh - 5 bytes
FE30h..FE41h - 18 bytes
FE43h..FE4Fh - 13 bytes

FE50h..FE53h - 4 bytes
FE55h - 1 byte
FE57h..FE5Fh - 9 bytes
FE60h..FE7Fh - 1 byte
FE61h..FE68h - 8 bytes
FE70h..FE7Dh - 14 bytes
FE7Fh - 1 byte (FFh, but this one is write-able)
FE80h - 1 byte
FE81h - 1 byte (FFh)
FE82h - 1 byte
FE8Ah..FE8Bh - 2 bytes
FE8Dh..FE8Fh - 3 bytes
FEA2h - 1 byte
FEB2h - 1 byte
FEB4h - 1 byte
FEB6h..FEB9h - 4 bytes
FEBBh..FEC4h - 10 bytes
FEC8h - 1 byte
FECAh - 1 byte
FECCh..FECDh - 2 bytes
FED1h..FED4h - 4 bytes
FED6h - 1 byte
FEDAh - 1 byte

These registers contain some control stuff. The firmware doesn't use them. All registers are 8bit R/W, except some MSBs aren't R/W:

```
FE1Dh.bit7:   NOT RW
FE1Eh.bit7:   NOT RW
FE1Fh.bit7:   NOT RW
FE39h.bit4-7: NOT RW
FE3Fh.bit1-7: NOT RW
FE4Bh.bit7:   NOT RW
FE4Dh.bit3-7: NOT RW
FE4Fh.bit7:   NOT RW
FE8Bh.bit7:   NOT RW
FE8Dh.bit3-7: NOT RW
FE8Eh.bit5-7: NOT RW
FEA2h.bit3-7: NOT RW
FEB2h.bit7:   NOT RW
FEB6h.bit5-7: NOT RW
FEB7h.bit1-7: NOT RW
FED1h.bit7:   NOT RW
FED3h.bit2-7: NOT RW
```

In most cases, changing the initial bits doesn't have visible effects, but there are still a lot of bits that do have visible effects:

```
  FE00h.bit4   Vertical.boldness
  FE00h.bit5   Temporarily NoSignal
  FE00h.bit7   Force_NTSC_Colors?
  FE0Dh:bit7   Untoggle=BlinksAvWhiteRecalib?
  FE0Fh.bit0-2 Change=TemporarilyNoSignal
  FE0Fh.bit3   ForcePALcolors_ButYetPAL60getsWrongYloc
                 (similar as FE01h.bit0, but HERE effect is inverse if
                 FE00h.bit7=x --> then forces NTSC colors)
  FE11h.bit2   ShortColorRollThenShowDitheredBlackWhiteAvImage
  FE11h.bit4-7 ShortColorRollThenShowDitheredBlackWhiteAvImage
  FE17h.bit1   Boldness
  FE17h.bit4-5 BoldnessWithContrast
  FE17h.bit6-7 PermanentlyToggleColoredAndDitheredBlackWhiteLines
  FE18h.bit1-2 BoldnessAndRightScreenEdge
  FE18h.bit6-7 PermanentlyToggleColoredAndDitheredBlackWhiteLines
  FE1Dh.bit6   NoSignalOrWeirdColorRoll
  FE1Eh.bit6   NoSignalOrWeirdColorRoll
  FE2Dh.bit7   ShortColorRollThenBlackWhiteDither
  FE31h.bit7   NoSignal
  FE3Ch.bit5   Boldness
  FE3Ch.bit6   BoldnessAndWashyBackground(SnesLooksAsWashyAsC64)
  FE3Ch.bit7   BoldnessBrighterBlack
  FE3Dh.bit6   UltraWashyBlurr
  FE3Dh.bit7   RollingImageAndBlackWhiteDither
  FE41h.bit6   TrueSnow?(not the artifical snow effect)
  FE41h.bit7   freezeSNOW (stuck BG or WHITE) and/or snow with wandering bars
  FE41h.bit6-7 C64:vague picture/sync traces? (uh, was this FE41h, not Fx41h?)
  FE43h        Somewhat Lumafactor/Lumaoffset for white-pixels getting
               brighter, or, if bit7 changed: white pixels are becoming BLACK
  FE44h.bit5   LessBoldness
  FE44h.bit6   AllMuchBrighter(including brighter black)
  FE44h.bit7   AllVERYMuchBrighter(including brighter black)
  FE45h.bit7   MonochromeImage(not dithered), with Washyness if C64 on AV2
  FE47h.bit7   ReddishMagentaInsteadRed, with Washyness if C64 on AV2
  FE48h.bit6   BlackWhiteDitheredImage
  FE4Eh.bit6   C64: Untoggling FE4Eh.6 shows C64 image for short moment
               (no effect on SNES image)
  FE50h.bit6-7 C64: does show C64 image going on/off (no effect on SNES)
  FE55h.bit4   AlmostCompletelyOmitsSomePalScanlines(instead of resampling
               them from PAL to 240pix-LCD-resolution)
  FE55h.bit5   SameAsBit4(but affecting OTHER lines)
  FE60h.bit1   When changed: ShortlyFlashesOrShortlyNoSignal
  FE60h.bit3   SNES: dims white to BluishGray and/or shortly NoSignal
  FE60h.bit3   C64: rolling white stripes when C64 ON
  FE80h.bit7   ForcesBlackImage
  FEB4h.bit7   Maybe S-Video (picks Snes/Luma from AV1, but C64/Chroma from
               AV2, showing hatchy blue borders at wrong hsync)
  FEB7h.bit0   Moves image 1pix LEFT
  FEC8h.bit0   DitheredBlackWhite
  FEC8h.bit7   Move image ca.80pix RIGHT and ca.12pix UP
  FECAh.bit0   UntogglingShortlyDimsBrightness
  FECAh.bit1   UntogglingShortlyBlinksWhitePixels
  FED1h.bit5-6 UponChanges: ShortlyNoSignal
  FED4h.bit1   OverBright (andWashyIfC64 is powered on AV2)
  FED6h.bit3   Snowy random Red/Magenta pixels (instead of SNES with Red
               background) (and if C64 powered on AV2: diagonally striped
               Red/Magenta) (might be also S-Video, chroma from AV2?)
  FED6h.bit7   ShortlyNoSignal, then UberOverBrightImage (Washy if C64 on AV2)
  FED6h.bit7   C64: does show C64 image, only once and then, goes on and off
  ;C64: FF20h.6-7: C64/SNOW color  ;XXX why FFxxh? THIS is FExxh!!!
  :     (also, FE20h would be read-only)
  ;C64: FF4Ah.7/FF4Bh.7/C64: bluish   ;XXX why FF4xh? THIS is FE4xh!!!

 ____________________________ AV Unused Registers ____________________________
```

FE69h..FE6Fh - Unknown/unused, 7 bytes ;(FFh-filled) ;NOT R/W
FEECh..FEEFh - Unknown/unused, 4 bytes ;(FFh-filled) ;NOT R/W
FEFEh - Unknown/unused, 1 byte ;(FFh-filled) ;NOT R/W
Unknown, not R/W, reading always seems to return FFh (but no open-bus garbage).

FEFFh - Unused (open-bus) (1 byte)
Probably unused I/O addresses.

AMT630A - FFxxh - LCD Registers (gamma/brightness/etc and IR)

FFD3h - IO_LCD_basic_contrast - Contrast (medium=7Eh)
FFD4h - IO_LCD_basic_brightness - Brightness (medium=8Eh)
FFD5h - IO_LCD_basic_tint - Tint (for NTSC only) (medium=00h)
FFD6h - IO_LCD_basic_saturation - Saturation (amount of color) (medium=38h)
The firmware user interface allows to configure these to Contrast=7Eh+(-28h..+28h), Brightness=8Eh+(-28h..+28h), Saturation=38h+(-28h..+28h), and, a bit more weirdly, Tint=00h+(93h..80h,00h,7Fh..62h), aka "Tint=(+13h..-1Eh), if Tint<>00h then Tint=Tint XOR 80h".
BUG: Setting Tint.bit7 does smash PAL color decoding (although Tint should affect NTSC only, and although the firmware doesn't even allow to change Tint when receiving a PAL signal; meaning that one could repair faulty PAL settings only when having a NTSC video source).
Note: For whatever reason, the firmware is reading the medium values from a table, with two separate columns for PAL and NTSC (both containing the above mentioned values: 7Eh,8Eh,00h,38h), plus six unused columns (which are all having these values: 80h,80h,00h,55h). Moreover, newer firmware changed the PAL/NTSC values to 80h,80h,00h,30h (and the six unused ones to 80h,80h,00h,36h).

FF01h..FF1Fh - IO_LCD_gamma_ramp_red (31 bytes)

FF20h..FF3Eh - IO_LCD_gamma_ramp_green (31 bytes)
FF3Fh..FF5Dh - IO_LCD_gamma_ramp_blue (31 bytes)

The firmware initializes all three ramps to the following non-linearily increasing values, starting with small steps (eg. +3 at 03h,06h), then having bigger steps in the middle (eg. +0Eh at 73h,81h), ending with small steps (eg. +4 at F6h,FAh), and with bigger final gap (+5/+6 from FAh to FFh/100h).

```
  03h,06h,0Ah,0Eh,14h,1Ah,21h,29h,34h,40h,4Dh,59h,66h,73h,81h,8Eh,
  9Ch,A7h,B1h,BAh,C2h,CAh,D0h,D7h,DDh,E2h,E7h,ECh,F1h,F6h,FAh
```

The newer firmware is using slightly different values (for 4.3" display):

```
  03h,07h,0Bh,10h,15h,1Bh,22h,2Ah,34h,3Fh,4Bh,58h,65h,72h,7Eh,89h
  96h,A2h,AEh,BAh,C4h,CDh,D5h,DCh,E2h,E7h,ECh,F0h,F4h,F8h,FBh
```

The power-up/reset values are yet different:

```
  07h,10h,1Ah,23h,2Dh,35h,3Ch,43h,4Ah,51h,56h,5Ch,62h,68h,6Eh,74h
  7Ah,81h,88h,8Fh,96h,9Dh,A4h,ABh,B2h,BAh,C1h,C9h,D2h,DDh,ECh
```

Note that the tables hold only 31 values, whilst internally, they are 33 entries wide (with fixed values 00h for Darkest, and FFh or 100h for Brightest intensities; accordingly, Darkest/Brightest intensities aren't affected by the Gamma settings) (OSD is often using dark/bright colors like Black, Cyan, White, Yellow - which won't get affected, whilst stuff like Gray or Brown would be affected by gamma - even on OSD layer) (also mind the odd OSD itensities (when IO_OSD_bright_transp_level is having too large brightness settings): eg. RGB values CCCh and FFFh both being White, and thus both being unaffected by gamma).

Purpose of the gamma ramp stuff is unknown: It might be intended to compensate incoming non-linear AV signals and/or to produce outgoing non-linear TFT signals. If it's aimed at AV then it's accidentally also applied to OSD. If it's aimed at TFT then it's apparently done wrong (because OSD colors look better with linear ramps) (not to mention that my firmware was compiled for a LD035H3 display whilst actually being shipped with a TM035KDH03 display).

FFCEh - IO_LCD_backdrop_color_Y (unsigned 8bit)
FFCFh - IO_LCD_backdrop_color_Cb (unsigned 8bit)
FFD0h - IO_LCD_backdrop_color_Cr (unsigned 8bit)

Seems to define the backdrop color in YCbCr (aka YUV) format. Common values are 13h,DDh,72h (Blue), and 00h,80h,80h (Black). The backdrop is shown when AV signal isn't present, optionally with Snow-effect.

Uh, the YCbCr theory doesn't seem to be quite right: Changing Y doesn't have any effect. And, changing MSBs of Cb/Cr does somewhat produce the expected colors... but unexpectedly having them applied to snow (eg. producing brown snow on black background, instead of white snow on brown background).

FFDAh - IO_backdrop_snow_level

The firmware sets this register to 6Ch, with Black backdrop (though a combination that looks neater would be 78h, with Blue backdrop).

```
  0-3   Snow Amount  (00h,01h=50% White, 06h=ManyWhite, 08h=FewWhite, etc.)
  4-6   Snow Width   (00h-07h = 1-8 pixels)
  7     ???
```

The snow color seems to be fixed (always White). However, it is affected by gamma-ramps, eg. setting blue/green ramps to all zeroes tweaks Red snow pixels (meaning that Snow is apparently actually VeryBrightGray, because White wouldn't be affected by the gamma ramps).

FFB0h - IO_LCD_snow_enable_and_misc

The firmware sets this to 22h, and changes bit7 for snow enable/disable, and alongsides rewrites bit5=1 for whatever reason.

```
  0     ???  affects AV picture NTSC? bright?
  1-4   ???
  5     ???  should be 1 (weirdly firmware often rewrites it as so)
  6     ???
  7     Snow Enable (0=Off, 1=On)
```

C64: bit5,7?=bluish (and bit5: OSD position/yscale depending on C64 on/off?)

FFB1h - IO_LCD_sharpness_or_so

```
  bit0    AV bolder/brighter/smeared pixels
  bit1-2  also bolder?
  bit3    VeryBlurry
  bit4,6  C64 becomes bluish?
```

FFCBh - IO_LCD_whatever_FFCBh

The firmware sets this register to 80h and 2Ah.

```
  0     Darker OSD colors?
  1-7   ???
```

FFD8h - IO_LCD_whatever_FFD8h

```
  0-6   ???
  7     ??? (firmware sets/clears this bit)
```

FFD2h - IO_LCD_forced_blank_color

Forced Blank Color (set to 4Fh..55h depending on Settings, and on display on/off) (4Fh=show AV video (or backdrop/snow), 5xh=force blank screen with fixed color) (does NOT affect OSD, affects only AV+backdrop+snow)

```
  for AV:
  0     Swap Red/Blue
  1     Mess
  2     Dim?
  3     AV_OFF/BLACK (show OSD only)
  7     C64: bluish on C64
```

FF00h - IO_LCD_config_FF00h - should be set to 03h
FFB2h - IO_LCD_config_FFB2h - should be set to 20h
FFB3h - IO_LCD_config_FFB3h - should be set to 20h
FFB4h - IO_LCD_config_FFB4h - should be set to 20h
FFB7h - IO_LCD_config_FFB7h - should be set to 90h (done twice)
FFCCh - IO_LCD_config_FFCCh - should be set to 80h
FFCDh - IO_LCD_config_FFCDh - should be set to 2Dh
FFD7h - IO_LCD_config_FFD7h - should be set to 10h

The firmware initializes these registers to fixed values. Changing bits seems to have no visible effect (except those listed below).

```
  FF00h.bit0/1: Affect AV color/brightness or so, maybe GammaRampMode/Enable?
  FFCCh.bit6:   Forces whole screen Mintgreen? (disables AV and OSD)
  FFCCh.bit7:   Brighter black
```

FFF0h - IO_LCD_config_FFF0h - should be set to 1Ah
FFF1h - IO_LCD_config_FFF1h - should be set to 06h
FFF2h - IO_LCD_config_FFF2h - should be set to D4h
FFF3h - IO_LCD_config_FFF3h - should be set to D2h
FFF4h - IO_LCD_config_FFF4h - should be set to F1h
FFF5h - IO_LCD_config_FFF5h - should be set to 0Eh
FFF6h - IO_LCD_config_FFF6h - should be set to 15h
FFF7h - IO_LCD_config_FFF7h - should be set to E4h
FFF8h - IO_LCD_config_FFF8h - should be set to F6h
FFF9h - IO_LCD_config_FFF9h - should be set to F1h
FFFAh - IO_LCD_config_FFFAh - should be set to 1Bh
FFFBh - IO_LCD_config_FFFBh - should be set to 81h

Related to YCbCr to RGB conversion (affects AV and and backdrop/snow, but doesn't affect OSD colors). The firmware initializes these registers to fixed values.

```
  Power-Up/Reset: 65h,C0h,DAh,0Dh,3Dh,19h,DAh,CDh,1Ah,3Dh,19h,81h
  Older firmware: 1Ah,06h,D4h,D2h,F1h,0Eh,15h,E4h,F6h,F1h,1Bh,81h
  Newer firmware: 11h,00h,00h,E9h,E1h,0Eh,09h,EEh,F4h,F1h,23h,81h
```

Unknown what the separate bytes are doing... maybe analog sensitivity settings or multipliers/dividers/offsets for YCbCr to RGB maths. The above old/new values are giving slightly different colors, whilst using other values can totally screw up the AV colors.

FFB5h..FFB6h - whatever, 02h bytes (unused by firmware)
FFB8h..FFBFh - whatever, 08h bytes (unused by firmware)
FFC0h..FFCAh - whatever, 0bh bytes (unused by firmware)
FFDBh..FFDCh - whatever, 02h bytes (unused by firmware)
FFDEh..FFEAh - whatever, 0dh bytes (unused by firmware)
FFD1h - whatever, 1 byte (unused by firmware)
FFD9h - whatever, 1 byte (unused by firmware)

The firmware doesn't use or initialize these registers. The registers are 8bit R/W, changing bits seems to have no visible effect (except for the two BrighterBlack bits).

```
  FFB8h.bit7   ;-changing this bit causes BrighterBlack
  FFD1h.bit6   ;-changing this bit causes BrighterBlack
```

FF5Eh - IO_LCD_whatever_FFh_1 (FFh)
FF5Fh - IO_LCD_whatever_FFh_2 (FFh)
FF60h - IO_LCD_whatever_FFh_3 (FFh)

Unknown, no visible effect when changing bits in these registers. Older firmware is leaving these registers unused, but newer firmware is setting them to FFh alongsides with LCD initialization (though the registers are FFh on power-up anyways).

```
 ________________________ Infrared IR REMOTE Registers ________________________
```

The AMT630A is said to support NEC and Philips RC-5 protocols. Existing/known firmwares are either leaving IR unsupported, or do merely implement the kinda useless NEC protocol (and without actually coming bundled with any IR sensor/hardware, neither on the mainboard, nor in the TFT screen unit).
The RC-5 protocol is a fairly well standarized protocol (eg. RC-5 based TV Remote Controls will be working with all RC-5 based TV Sets).
The NEC protocol is shared by numerous japanese companies, each having different maker/device IDs assigned, and each using different command values (eg. NEC based TV Remote Controls will be incompatible with almost all NEC based TV Sets, unless the remote control contains very huge databases for different TV sets; or unless the TV Set supports some learning function, such as prompting the user to press the button that is to be used as Standby button).

FF83h - IO_IR_stat_FF83h_used (R) ;lsb (firmware supports 00h or 86h)
FF85h - IO_IR_stat_FF85h_used (R) ;msb (firmware supports FFh or 6Bh)
These registers seem to contain the received NEC device ID. In original NEC protocol, the second byte would be just the inverse of the other byte (XOR FFh). In later NEC extensions, the second byte can have other values.
The received NEC command (and some flags) are found in SFR area (SFR_IO_IR_data & SFR_IO_IR_flags). And, External Interrupt 0 is used for IR.

FF61h - IO_IR_config_FF61h ;=05h ;\
FF62h - IO_IR_config_FF62h ;=3Fh ;
FF63h - IO_IR_config_FF63h ;=10h ;
FF64h - IO_IR_config_FF64h ;=04h ; ;<--bit3-7: NOT RW
FF65h - IO_IR_config_FF65h ;=76h ;
FF66h - IO_IR_config_FF66h ;=3Bh ;
FF67h - IO_IR_config_FF67h ;=08h ;
FF68h - IO_IR_config_FF68h ;=15h ;
FF69h - IO_IR_config_FF69h ;=07h ;
FF6Ah - IO_IR_config_FF6Ah ;=00h ;
FF6Bh - IO_IR_config_FF6Bh ;=00h ; ;<--NOT RW (but is written!?!)
FF6Ch - IO_IR_config_FF6Ch ;=00h ; ;<--NOT RW (but is written!?!)
FF6Dh - IO_IR_config_FF6Dh ;=00h ; ;<--NOT RW (but is written!?!)
FF6Eh - IO_IR_config_FF6Eh ;=00h ; ;<--NOT RW (but is written!?!)
FF6Fh - IO_IR_config_FF6Fh ;=76h ;
FF70h - IO_IR_config_FF70h ;=1Ch ;
FF71h - IO_IR_config_FF71h ;=08h ;
FF72h - IO_IR_config_FF72h ;=F5h ;/
FF77h - IO_IR_config_FF77h ;=59h ;\
FF78h - IO_IR_config_FF78h ;=00h ;
FF79h - IO_IR_config_FF79h ;=01h ;
FF7Ah - IO_IR_config_FF7Ah ;=73h ;
FF7Bh - IO_IR_config_FF7Bh ;=FFh ;
FF7Ch - IO_IR_config_FF7Ch ;=02h ;
FF7Dh - IO_IR_config_FF7Dh ;=0Fh ;/

FF82h - IO_IR_config_FF82h ;=01h ;- ;<--bit4-7: NOT RW
Initialized to above fixed values for NEC protocol (unknown how to use/initialize RC-5 protocol).
Alongsides, SFR_IO_PORT0_MODE_A/B.bit3 should be configured before using IR.

FF73h..FF76h - probably IR related, 4 bytes
FF7Eh..FF81h - probably IR related, 4 bytes
FF84h - probably IR related, 1 byte (not R/W)
Unknown, not used by firmware. Probably IR related (as they are within the IR register area).

FF86h - maybe IR related, 1 byte (not R/W)
FF87h - maybe IR related, 1 byte (not R/W)
FF88h..FF8Bh - maybe IR related, 4 bytes
FF8Ch - maybe IR related, 1 byte (bit6-7: not R/W)
FF8Dh..FF91h - maybe IR related, 5 bytes (not R/W)
FF92h..FF9Eh - maybe IR related, 13 bytes
FF9Fh - maybe IR related, 1 byte (bit6-7: not R/W)
FFA0h..FFA3h - maybe IR related, 4 bytes
Unknown, not used by firmware. Maybe IR related (as they are located at the end of IR register area). Or maybe LCD related, or something completely different. There's no visible effect when changing bits in these registers.

```
 _____________________________ LCD Unused Registers _____________________________
```

FFA4h..FFAFh - Unused (open-bus) (0Ch bytes)
FFEBh..FFEFh - Unused (open-bus) (05h bytes)
FFFCh..FFFFh - Unused (open-bus) (04h bytes)
FFDDh - Unused (open-bus) (01h bytes)
Probably unused I/O addresses.

AMT630A - Component Lists & Pinouts

Component List for 3.5 inch display (nocash)

```
  mainboard: "ZCD630A-3.5D_V1.1"
  main chip: 64pin AMT630A (video controller with 8031 cpu)
  firmware: 8pin "MK, 25D40BTIG, 1720" (512kbyte spi flash, D=DualDataPin)
  voltage regulator 1:  8pin "XL1509" (pin1=Vin/12V, pin2=Vout/5V)
  voltage regulator 2:  3pin (5V to 3.3V)
  backlight driver: 6pin "7001" (Micrel MIC3287 or compatible)
  connectors: 54pin display, 2pin keypad, 4pin video/supply, 4pin sio/unused
  display: "RoHS 1580005880 111020, TM035KDH03, 76DK14A04A1 OMP 11111009, 1"
```

Component List for 5.0 inch display (bigby)

```
  mainboard: "BO-39-Y-7795BH1" ?
  main chip: 64pin "AMT630A, G171800030"
  firmware spi flash: "MK, xxD040BTIG, S782A1"
  voltage regulator: "RZC2013S, SZ82"
  backlight driver "D2AxD ?"
  connectors: 40pin display, 2pin keypad, 4pin video/supply, 4pin sio/unused
  display "GP Innolux Display, AT050TN33 V.1 AA0500004101, S/N: 895W 001-000WB"
  crystal "X27.000"
```

note: firmware seems to be for 4.3 inch (probably AT043TN25, which has same resolution as 5.0 inch AT050TN33)

Inches, Ratios, Resolutions

Below are some common dimensions for small/medium sized TFT screens. For internet search engines it's best to enter the size and ratio, eg. "3.5" AND "4:3" (that's reducing the likelyness of ratio "4:3" being mistreated as "4.3" inches).

```
  3.5 inch   4:3  320x240
  4.3 inch  16:9  480x272
  5.0 inch  16:9  480x272 or 800x480
  5.6 inch   4:3  1024x768 or 320x234? or 800x600?
  7.0 inch  16:9  1024x600 or 480x234?
  8.0 inch   4:3  1024x768
```

And,

```
  AV-to-USB video grabbers
```

AMT630A chips are likely to be found in cheap displays (3.5", 4.3", 5.0" with resolution 320x240 or 480x272) for around $15-$25. AMT630A chips can be also found in AV-to-USB video grabbers (with an additional USB chip, instead of the TFT display).
Theoretically, AMT630A does also support bigger displays with resolutions up to 1024x768, however, displays with higher resolutions and/or more than 5.0" are usually more expensive (around $40), and they are likely to contain more advanced video controllers (with RGB/VGA/HDMI video inputs).

AMT630A (64pin LQFP package)

```
 Pin TYPE Function
  1  A CVBS1                                    ;-external video YELLOW
  2  A CVBS2                                    ;-GNDed
  3  A CVBS3                                    ;-external video WHITE
  4  A VCOM_ADC                                 ;-
  5  P AVSS_ADC (GND)
  6  D P03    REMOTE                            ;-
  7  A P00    SAR2
  8  A P01    SAR1
  9  A P02    SAR0                              ;-Keypad
  10 P DVDD 3.3V
  11 P GND
  12 P VDD 1.2v (build-in LDO for 1.2v core power)
  13 P AGND (AVSS33_ANA)
  14 A XTAL_OUT                                 ;\27MHz
  15 A XTAL_IN                                  ;/
```

```
16 P AVDD 3.3V
---
17 D P10    SPI_CS                                 ;\
18 D P11    SPI_SI                                 ; SPI FLASH
19 D P12    SPI_SO                                 ;
20 D P13    SPI_CLK                                ;/
21 P DVDD 3.3V
22 P VDD 1.2v (build-in LDO for 1.2v core power)
23 P GND
24 D P14    cpu_rstn R0  VOS tcon_r0  ituVDE   sVSY   ;\LCD.   D
25 D P15    cpu_cs   R1  HOS tcon_r1  ituHDE   sHSY   ; LCD.   D
26 D P16    cpu_rs   R2  DOE tcon_r2  ituDEO   sDEN   ; LCD.   D
27 D P17    cpu_wr   R3  DCK tcon_r3  ituclko  sCLK   ; LCD.   D
28 D P20    cpu_rd   R4  B7  tcon_r4  itu_d7   sD7    ; LCD.   D
29 D P21    cpu_d17  R5  B6  tcon_r5  itu_d6   sD6    ; LCD.   D
30 D P22    cpu_d16  R6  B5  STVR     itu_d5   sD5    ; LCD.   D
31 D GPIO0  cpu_d15  R7  B4  STVL     itu_d4   sD4    ;/LCD.   D   (blue.7)
32 D GPIO1  cpu_d14  G0  B3  tcon_g0  itu_d3   sD3    ;\LCD.   D
---
33 D GPIO2  cpu_d13  G1  B2  tcon_g1  itu_d2   sD2    ; LCD.   D
34 D GPIO3  cpu_d12  G2  B1  tcon_g2  itu_d1   sD1    ; LCD.   D
35 D GPIO4  cpu_d11  G3  B0  tcon_g3  itu_d0   sD0    ; LCD.   D
36 D P23    cpu_d10  G4  G7  tcon_g4                  ; LCD.   D
37 D P24    cpu_d9   G5  G6  tcon_g5                  ; LCD.   D
38 D P25    cpu_d8   G6  G5  CKV                      ; LCD.   D
39 D P26    cpu_d7   G7  G4  OEV                      ;/LCD.   D   (green.7)
40 P DVDD 3.3V
41 P VDD 1.2v (build-in LDO for 1.2v core power)
42 P GND
43 D GPIO5  cpu_d6   B0  G3  tcon_b0  itu_d0'  sD0'   ;\LCD.   D
44 D GPIO6  cpu_d5   B1  G2  tcon_b1  itu_d1'  sD1'   ; LCD.   D
45 D GPIO7  cpu_d4   B2  G1  tcon_b2  itu_d2'  sD2'   ; LCD.   D
46 D GPIO8  cpu_d3   B3  G0  tcon_b3  itu_d3'  sD3'   ; LCD.   D
47 D GPIO9  cpu_d2   B4  R7  tcon_b4  itu_d4'  sD4'   ; LCD.   D
48 D GPIO10 cpu_d1   B5  R6  tcon_b5  itu_d5'  sD5'   ; LCD.   D
---
49 D P27    cpu_d0   B6  R5  STHR     itu_d6'  sD6'   ; LCD.   D
50 D P30    cpu_rd   B7  R4  POL      itu_d7'  sD7'   ;/LCD.35 D23 (red.7)
51 D P31    cpu_wr   DCK R3  tcon_clk ituclko' sCLK'  ;\LCD.38 CLK
52 D P32    cpu_rs   DOE R2  STHL     ituDEO'  sDEN'  ; LCD.52 DEN
53 D P33    cpu_cs   HOS R1  OEH      ituHDE'  sHSY'  ; LCD.36 HSYNC
54 D P34    cpu_rstn VOS R0  tck2     ituVDE'  sVSY'  ;/LCD.37 VSYNC
55 D P35    DC_PWM0                       ;backlight driver  (via 1K ohm)
56 D P36    DC_PWM1  TXD''                ;display spi.dta   (via 33 ohm)
57 D P37    DC_PWM2  RXD''                ;display spi.clk   (via 33 ohm)
58 D P07    DC_PWM3  RXD'                 ;display spi.cs    (via 33 ohm)
59 D P06    DC_PWM2' TXD'                 ;display spi.reset (via 33 ohm)
```

```
  60 D P04    SDA      TXD                ;to external connector
  61 D P05    SCL      RXD                ;to external connector
  62 P DVDD 3.3V
  63 D RESET                              ;-reset (active HIGH)
  64 P AVDD_ADC 3.3V
```

Firmware FLASH (8pin "MK, 25D40BTIG", 512kbyte spi flash, D=DualDataPin)

```
  1 /CS
  2 DO
  3 /WP (VCC'ed)
  4 GND
  5 DI
  6 CLK
  7 /HOLD (VCC'ed)
  8 VCC (3.27V) (even when video off)
```

Backlight driver "7001" (aka Microchip/Micrel MIC3287, package TSOT-23-6)

```
  1 SW  Switch Node ("Input"): Internal power bipolar collector
  2 GND Ground
  3 FB  Feedback (input): Output voltage sense node.
  4 EN  Enable (input): High=enable, Low=Shuts Down              ;PWM
  5 OVP Overvoltage Protection (Input): Connect to the output    ;LED Anode
  6 VIN Supply (input): 2.8V to 6.5V for internal circuitry      ;5V
```

Pin3 (FB): Connect the cathode of the LED to this pin. A resistor from this to ground sets the LED current.

Innolux TFT LCD Panel Driving Section

FPC Connector is used for the module electronics interface. The recommended model is FH19SC-40S-0.5SH manufactured by HIROSE.

```
  Pin Symbol I/O Function Remark
  1  VLED- P Power for LED backlight cathode    ;\backlight
  2  VLED+ P Power for LED backlight anode      ;/
  3  GND   P Power ground                       ;\supply
  4  VDD   P Power voltage                      ;/
  5  R0    I Red data (LSB)
  6  R1    I Red data
  7  R2    I Red data
  8  R3    I Red data
  9  R4    I Red data
  10 R5    I Red data
  11 R6    I Red data
  12 R7    I Red data (MSB)
  13 G0    I Green data (LSB)
  14 G1    I Green data
  15 G2    I Green data
  16 G3    I Green data
  17 G4    I Green data
```

```
18 G5    I Green data
19 G6    I Green data
20 G7    I Green data (MSB)
21 B0    I Blue data (LSB)
22 B1    I Blue data
23 B2    I Blue data
24 B3    I Blue data
25 B4    I Blue data
26 B5    I Blue data
27 B6    I Blue data
28 B7    I Blue data (MSB)
29 GND   P Power ground
30 CLK   I Pixel clock
31 DISP  I Display on/off
32 NC    - No Connection
33 NC    - No Connection
34 DE    I Data Enable
35 NC    - No Connection
36 GND   P Power ground
37 X1  I/O Right electrode - differential analog   ;\
38 Y1  I/O Bottom electrode - differential analog  ; touchpad
39 X2  I/O Left electrode - differential analog    ; (unused)
40 Y2  I/O Top electrode - differential analog     ;/
I: input, O: output, P: Power
```

Tianma Display Pinout (ribbon cable)

```
1 LED_Cathode ;\              ;\GND via 2.2 ohm
2 LED_Cathode ; backlight     ;/
3 LED_Anode   ;               ;\
4 LED_Anode   ;/              ;/
5 NC          ;\
6 NC          ; not connect
7 NC          ;/
8 RESET       ;-reset
9 SPENA       ;\
10 SPCK       ; SPI bus
11 SPDA       ;/
12 D00        ;\
13 D01        ;
14 D02        ;
15 D03        ; LCD
16 D04        ;
17 D05        ;
18 D06        ;
19 D07        ;/
20 D08        ;\
21 D09        ;
```

```
  22 D10        ;
  23 D11        ; LCD
  24 D12        ;
  25 D13        ;
  26 D14        ;
  27 D15        ;/
  28 D16        ;\
  29 D17        ;
  30 D18        ;
  31 D19        ; LCD Red (in default 24bit+HV mode)
  32 D20        ;
  33 D21        ;
  34 D22        ;
  35 D23        ;/
  36 HSYNC      ;\LCD Sync (in default 24bit+HV mode)
  37 VSYNC      ;/
  38 CLK        ;-data clock
  39 NC         ;\not connect
  40 NC         ;/
  41 VDD        ;\power 3.3V
  42 VDD        ;/
  43 NC         ;\
  44 NC         ;
  45 NC         ; not connect
  46 NC         ;
  47 NC         ;
  48 NC         ;
  49 NC         ;
  50 NC         ;
  51 NC         ;/
  52 DEN        ;-LCD data enable
  53 GND        ;\ground
  54 GND        ;/
```

SPI FLASH to Parallel Port

No$x51 includes a function for uploading binaries to SPI FLASH via parallel port, this is very useful for uploading/testing binaries on hardware (leaving apart that I don't know of anybody else owning a parallel port).

```
  AMT630A side                    PC Side (36pin Centronics or 25pin SUBD)
  DTA.W FLASH.pin5    ----[1K]---- D0    CNTR.pin2     SUBD.pin2
  CLK   FLASH.pin6    ----[1K]---- D1    CNTR.pin3     SUBD.pin3
  /CS   FLASH.pin1    ----[1K]---- D2    CNTR.pin4     SUBD.pin4
  DTA.R FLASH.pin2    --[74HCxx]-- BUSY  CNTR.pin11    SUBD.pin11
  RESET AMT630A.pin63 ----|<|----- /INIT CNTR.pin31    SUBD.pin16
  GND   SUPPLY.Black  ------------ GND   CNTR.pin19-30 SUBD.pin18-25
```

RESET is active HIGH, it's used to stop the 80C52 CPU and to prevent it accessing the SPI bus during uploads; the diode just prevents it from getting pulled LOW when not issuing reset.

D0,D1,D2 are SPI outputs from PC, the 1K resistors NEEDED (else AMT630A won't work even when PC data lines are HighZ).
BUSY is SPI input on PC side, passed through some noninvering 74HCxx AND/OR gate (else FLASH chip freaks out about 5V pullup from PC).
The above circuit is far from perfection. I've tested it with two parallel ports, one didn't work, the other does work, although it tends to reply with a wrong FLASH chip ID after booting the PC - but that can be fixed by unplugging the AMT630A power supply for a moment.
When designing a better voltage conversion circuit, mind that PC outputs need to be switched off after upload, so that the AMT630A can access the SPI bus once when RESET is released.
No$x51 software notes: Select the LPT port in no$x51 setup. Load a binary (or assemble some source code), then use the file/upload function. Upload should work with known FLASH chip IDs (such as MK 25D40), but may fail if you have another/unknown FLASH chip.

Index

[extracted from no$x51 v1.5 documentation]