

1. Data Protocol

A. Display data protocol

- command nibble (to FTDD006): '1 b c d'

b,c,d are dimming coefficients:

0,0,0 is maximum brightness.

1,1,1 is minimum brightness (display completely off).

Note: - FTDD006 interpretes any coefficient different from 1,1,1 as 0,0,0 (maximum brightness).

- At maximum brightness, the DISOFF output = 0, at minimum brightness, the DISOFF output = 1.

- In the initial state, all grids and segments are off, and maximum brightness (DISOFF = 0).

- segment nibbles (to FTDD006): 'a b c d'

FTDD006 is capable of driving a display of 8 grids by 12 segments (if input NORMSEG = '1') or 8 grids by 20 segments (if NORMSEG = '0').

For the 8 * 12 configuration, three consecutive nibbles form the complete info for 1 grid, and all grids are sent in order.

segments	1 2 3 4	5 6 7 8	9 10 11 12
grid 1	1: a b c d	2: a b c d	3: a b c d
grid 2	4: a b c d	5: a b c d	6: a b c d
⋮			
grid 8	22: a b c d	23: a b c d	24: a b c d

For the 8 * 20 configuration, five consecutive nibbles form the complete info for 1 grid, and all grids are sent in order.

segments	1 2 3 4	5 6 7 8	9 10 11 12	13 14 15 16	17 18 19 20
grid 1	1: a b c d	2: a b c d	3: a b c d	4: a b c d	5: a b c d
grid 2	6: a b c d	7: a b c d	8: a b c d	9: a b c d	10: a b c d
⋮					
grid 8	36: a b c d	37: a b c d	38: a b c d	39: a b c d	40: a b c d

Nibbles exceeding the display capacity are ignored.

Bit = 1 corresponds to segment on.

B. Keyboard data protocol

- command nibble (to FTDD006): '0 b c d'

b,c,d are selecting coefficients for the next nibble: 0,0,0 is: 'give driver state'
0,0,1 is: 'give driver ID'
others: reserved

- driver info :

- driver state (from FTDD006): 'a b c d'

driver state '1 0 1 0' corresponds to driver functioning properly.

- driver ID (from FTDD006): 'a b c d'

driver ID '0 1 0 1' corresponds to FTDD006.

- key nibbles (from FTDD006):

FTDD006 scans for 8 x 4 keys, nibble #1 corresponds to segment 1, nibble #2 corresponds to segment 2, etc. until nibble #8.
Bit = 1 corresponds to key in.

- Checksum (from FTDD006) : this is the 4-bit sum (ignoring overflows) on the driver info (driver state or driver ID) and all key nibbles.

- RCS nibbles (not implemented in FTDD005): 'a b c d'

1:	startbit(1)	startbit(2)	togglebit	system(5)
2:	system(3)	system(2)	system(1)	system(0)
3:	0	0	command(5)	command(4)
4:	command(3)	command(2)	command(1)	command(0)

- Checksum (not implemented in FTDD006) : this is the 4-bit sum (ignoring overflows) on all RCS nibbles.

2. Bus Protocol

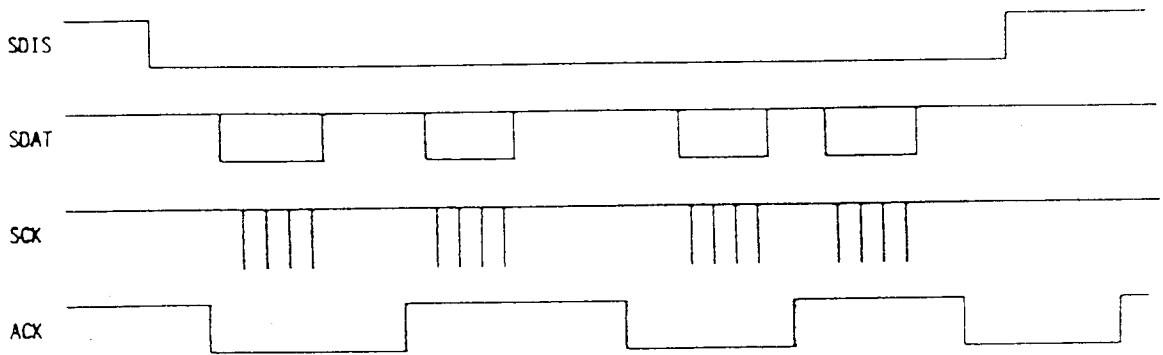
The interface is made with 5 signals:

- SDIS (Select DISplay driver): indicates the start (falling edge) and end (rising edge) of a communication sequence between FTDD006 and the master μ P. The communication sequence can be ended after an acknowledge (or an acknowledge time-out). SDIS is generated by the master μ P.
- ACK (ACKnowledge): indicates that FTDD006 is ready to send/receive the next nibble or after the termination of the communication sequence. ACK is generated by FTDD006 within max T.B.F. (300 ?) μ s. An acknowledge is indicated by the reversing of the ACK line. The initial state of the ACK line is low: this indicates that the driver is ready to receive displaydata and to send keyboard data.

- SI (Serial Input): serial data input line of FTDD006 (is data output of master μ P), LSB first.
- SO (Serial Output): serial data output line of FTDD006 (is data input of master μ P), LSB first.
- SCK (Serial Clock): serial clock line, is controlled by the master μ P. The rest state of the SCK line is high, data is shifted in/out by FTDD006 at the falling edge of the clock. Both low- and high-period of clocksignals are min 4 μ s. The interface is 4-bit, so clocksignals have to come by 4.

The serial interface is half-duplex, this means that the serial input or the serial output can be active, but never both at the same time. Because of this half-duplex interface, and both SI and SO are open-drain gates at FTDD006 as well as the master μ P, SI and SO can be tied together to form SDAT (Serial DATA), the rest-state has to be high.

Bus protocol example:



3. FTD Protocol

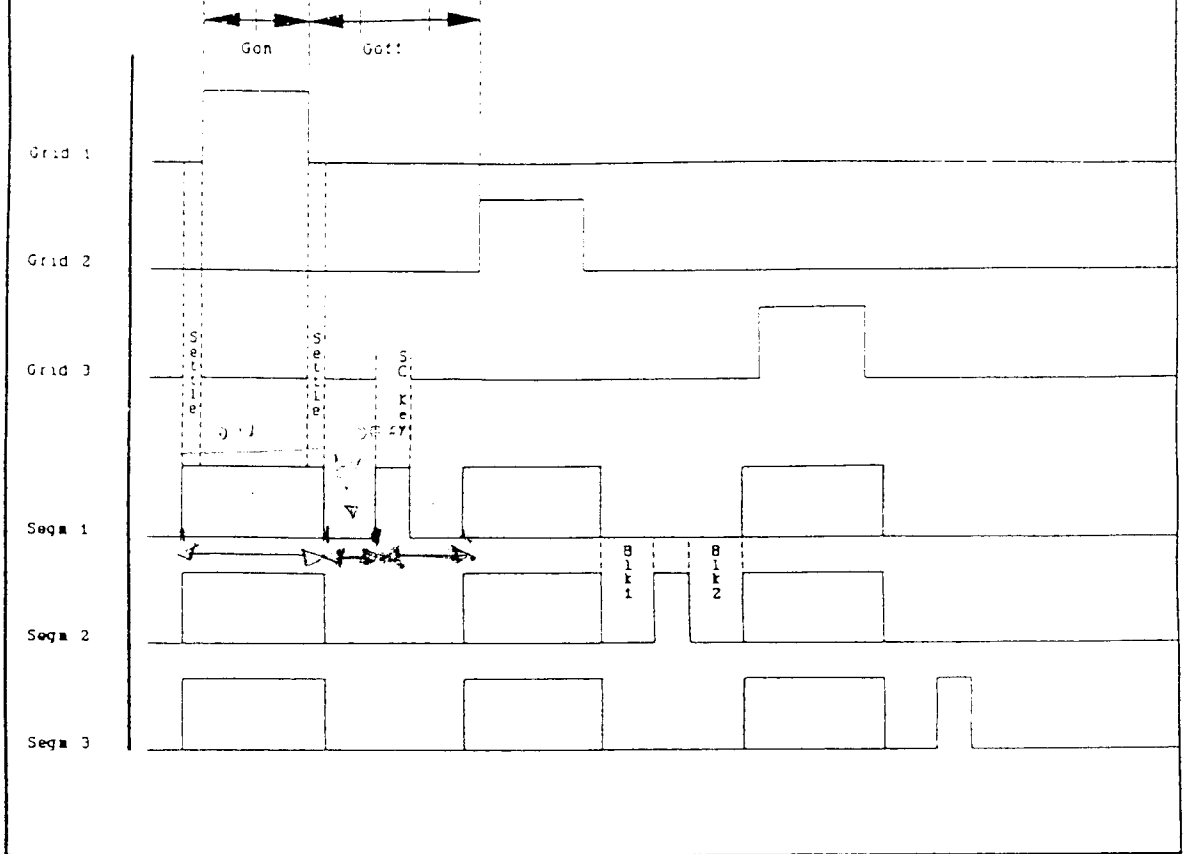
The FTD protocol describes the controlling of the FTD-display and the scanning of the keyboard.

If the display is set at minimum brightness, the FTD is not scanned and all gridlines are off. All keyboard scan lines are set active (silent scan). When a key is detected, all keyboard lines are scanned to read the pressed key(s).

During the scanning of the FTD, all 8 gridlines shall be activated in turn, starting with grid1, then grid2, etc... In 8 x 20 mode, segments 13 (sent first) up to 20 (sent last) have to be inverted to allow an inverting buffer.

The scanning of the keyboard is realised by controlling the segment-lines when the gridlines are inactive. This scanning is similar to the controlling of the FTD, but now segment1 upto segment8 will be activated in turn.

See figure on next page:



Settle: The time before a grid is activated while the segment lines are already activated, or the time that a grid is already inactive while the segments are still active.

Gon: The time that a grid is on (full brightness).
Gon is app. 1000 μ s for 8 x 12, and app. 800 μ s for 8 x 20.

Goff: The time that all grids are inactive.
Gon + Goff is app. 1250 μ s.

Blk1: Blanking time after segments go inactive and before keyscan.
Blk1 is min. 40 μ s.

Blk2: Blanking time after keyscan and before segments can be activated.
Blk2 is min. 40 μ s.

Sckey: The time that is needed by the processor to input the key lines. The time between the set-up of the scanline and the reading of the key line is min 20 μ s.
Sckey is app. 70 μ s.
