STM32F3 Microcontroller

GPIO Applications

27/02/2014

Dr. Cuauhtémoc Carbajal
Agenda

• 3V-5V interfacing
• STM32F3 Electrical Characteristics
• I/O Device Categories
• GPIO Interfacing to External Devices
The need for interfacing between 3V and 5V systems

- Many reasons exist to introduce 3V systems, notably the lower power consumption for mobile applications and the introduction of parts that use technologies with such fine geometries that 5V is simply not allowed any more.
- There is a gradual transition from 5V to 3V, since not always are all required components available, or the system is rather complex so that 3V is introduced in part of a system.
- Because many devices still are 5V devices, and most modern sensors, displays, and flash cards are 3V-only, many makers find that they need to perform level shifting/conversion.
- We obviously want a reliable signal transfer from the 5V system to the 3V system and vice versa. This implies that the output voltages should be such that the input levels are satisfied.
Interfaces Between a 3V Microcontroller and 5V Systems
Important characteristics are:

- $V_{IH\text{min}}$ min value input recognized as a ‘1’
- $V_{OH\text{min}}$ min value of output generated as a ‘1’
- $V_{IL\text{max}}$ max value of input recognized as a ‘0’
- $V_{OL\text{max}}$ max value of output generated as a ‘0’

- Values outside the given range are not allowed.
Digital circuits normally come in two versions:
- TTL levels: \( V_{IL} = 0.8V, V_{IH} = 2.0V \)
- CMOS levels: \( V_{IL} = 0.3*V_{CC}, V_{IH} = 0.7*V_{CC} \).
Level Shifting

• 5V to 3V
  – All 5V families have an output voltage swing that is large enough to drive 3V reliably. Outputs may be as high as 3.5V for many TTL output stages, to the full 5V for many CMOS outputs. Therefore, as far as switching levels are concerned, there are no problems in interfacing from 5V to a 3V system.

• 3V to 5V
  – All 3V logic families deliver practically the full output voltage swing of 3V, so they can drive TTL switching levels without problems.
  – However, a 3V system cannot reliably drive a 5V one that has CMOS input levels, even when using pull-up resistors.
Voltage level conversion methods

- Discrete
  - You can do voltage level conversion using discrete bipolar transistors.
  - The circuit shown converts from a voltage swing of 0-3V to a voltage swing of 0-5V.
  - The resistor values may have to be modified depending on the switching speed required. At the same time, the resistor connected to the collector should be as large as possible in low power applications, since static current will be drawn when the output from the circuit is low.
  - Also note that this circuit inverts, i.e. it will drive the output low when the input is high and vice versa.
  - You can use the same circuit with an NMOS, but in this case you do not need a resistor in series with the gate. Make sure that you select a transistor with an appropriate threshold voltage.

http://www.daycounter.com/Circuits/Level-Translators/Level-Translators.html
Voltage level conversion methods

• Passive voltage divider
  – If you are only concerned with avoiding violation of the absolute maximum ratings of the 3V circuit, you can use a resistor voltage divider to divide down the 5V signal to 3V. With an appropriate choice of resistors, this will work fine, but it will draw static current all the time.
  – Typical resistor values for the figure below can be $R_1=22\,\text{k}\Omega$, $R_2=33\,\text{k}\Omega$. If the 5V device has a low enough threshold voltage that it will function with a 3V input voltage, this can be a good approach for bi-directional signals, as the voltage divider only divides down the voltage in one direction.
  – Note: Can work on bi-directional signals if 5V system has low enough threshold voltage ($V_{IH5V}<V_{OH3V}$)
Voltage level conversion methods

- **Dual VCC level shifters**
  - The **74LVC4245 and 74ALVC164245** (8 and 16 bits resp.) are CMOS transceivers fed from both 3V and 5V supplies. The level shifting is done internally and the parts have full output voltage swings at both sides, making them ideal for level shifting purposes, especially when driving 5V CMOS levels.
  - Dual VCC level shifters are superior alternatives to the sometimes used input pull-up resistors, blocking diodes and other circuits that normally degrade speed and/or noise margins.
  - Only problem is that they are only good in one direction which can be a problem for some specialty bi-directional interfaces and also makes wiring a little hairy.

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>OE</td>
<td>DIR</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
</tr>
</tbody>
</table>

FUNCTION TABLE
Voltage level conversion methods

• Logic Level Converter BOB-08745
  – It safely steps down 5V signals to 3.3V and steps up 3.3V to 5V. This level converter also works with 2.8V and 1.8V devices. Each level converter has the capability of converting 4 pins on the high side to 4 pins on the low side. Two inputs and two outputs are provided for each side.
  – Can be used with normal serial, I2C, SPI, and any other digital signal. It does not work with an analog signal.

https://www.sparkfun.com/products/8745
Voltage level conversion methods

- 8-channel Bi-directional Logic Level Converter - TXB0108
  - This chip perform bidirectional level shifting from pretty much any voltage to any voltage and will **auto-detect** the direction. Only thing that doesn't work well with this chip is i2c (because it uses strong pullups which confuse auto-direction sensor).
  - If you need to use pullups, you can but they should be at least 50K ohm - the ones internal to the STM32F3 are about 100K ohm so those are OK!

http://www.adafruit.com/products/395
Voltage level conversion methods

• ULN2003A are high-voltage high-current Darlington transistor arrays. It consists of seven NPN Darlington pairs that feature high-voltage outputs with common-cathode clamp diodes for switching inductive loads.

• The collector-current rating of a single Darlington pair is 500 mA.

• The Darlington pairs can be paralleled for higher current capability.
Voltage level conversion methods

- ULN2003A applications include relay drivers, hammer drivers, lamp drivers, display drivers (LED and gas discharge), line drivers, and logic buffers.

Interfaces With High-Current Output Buffers ULN2003
Voltage level conversion methods

• Optocoupler
  – It can be used to control a circuit that is completely isolated from your uC.
  – In this case, imagine that the LED and battery pack are a hacked toy which you're turning on and off with the uC.

## STM32F3 I/O structure

<table>
<thead>
<tr>
<th>Pin</th>
<th>Structure</th>
<th>Pin</th>
<th>Structure</th>
<th>Pin</th>
<th>Structure</th>
<th>Pin</th>
<th>Structure</th>
<th>Pin</th>
<th>Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA0</td>
<td>Ta</td>
<td>PB0</td>
<td>Ta</td>
<td>PC0</td>
<td>Ta</td>
<td>PD0</td>
<td>FT</td>
<td>PE0</td>
<td>FT</td>
</tr>
<tr>
<td>PA1</td>
<td>Ta</td>
<td>PB1</td>
<td>Ta</td>
<td>PC1</td>
<td>Ta</td>
<td>PD1</td>
<td>FT</td>
<td>PE1</td>
<td>FT</td>
</tr>
<tr>
<td>PA2</td>
<td>Ta</td>
<td>PB2</td>
<td>Ta</td>
<td>PC2</td>
<td>Ta</td>
<td>PD2</td>
<td>FT</td>
<td>PE2</td>
<td>FT</td>
</tr>
<tr>
<td>PA3</td>
<td>Ta</td>
<td>PB3</td>
<td>FT</td>
<td>PC3</td>
<td>Ta</td>
<td>PD3</td>
<td>FT</td>
<td>PE3</td>
<td>FT</td>
</tr>
<tr>
<td>PA4</td>
<td>Ta</td>
<td>PB4</td>
<td>FT</td>
<td>PC4</td>
<td>Ta</td>
<td>PD4</td>
<td>FT</td>
<td>PE4</td>
<td>FT</td>
</tr>
<tr>
<td>PA5</td>
<td>Ta</td>
<td>PB5</td>
<td>FT</td>
<td>PC5</td>
<td>Ta</td>
<td>PD5</td>
<td>FT</td>
<td>PE5</td>
<td>FT</td>
</tr>
<tr>
<td>PA6</td>
<td>Ta</td>
<td>PB6</td>
<td>FTf</td>
<td>PC6</td>
<td>FT</td>
<td>PD6</td>
<td>FT</td>
<td>PE6</td>
<td>FT</td>
</tr>
<tr>
<td>PA7</td>
<td>Ta</td>
<td>PB7</td>
<td>FTf</td>
<td>PC7</td>
<td>FT</td>
<td>PD7</td>
<td>FT</td>
<td>PE7</td>
<td>Ta</td>
</tr>
<tr>
<td>PA8</td>
<td>FT</td>
<td>PB8</td>
<td>FTf</td>
<td>PC8</td>
<td>FT</td>
<td>PD8</td>
<td>Ta</td>
<td>PE8</td>
<td>Ta</td>
</tr>
<tr>
<td>PA9</td>
<td>FTf</td>
<td>PB9</td>
<td>FTf</td>
<td>PC9</td>
<td>FT</td>
<td>PD9</td>
<td>Ta</td>
<td>PE9</td>
<td>Ta</td>
</tr>
<tr>
<td>PA10</td>
<td>FTf</td>
<td>PB10</td>
<td>Ta</td>
<td>PC10</td>
<td>FT</td>
<td>PD10</td>
<td>Ta</td>
<td>PE10</td>
<td>Ta</td>
</tr>
<tr>
<td>PA11</td>
<td>FT</td>
<td>PB11</td>
<td>Ta</td>
<td>PC11</td>
<td>FT</td>
<td>PD11</td>
<td>Ta</td>
<td>PE11</td>
<td>Ta</td>
</tr>
<tr>
<td>PA12</td>
<td>FT</td>
<td>PB12</td>
<td>Ta</td>
<td>PC12</td>
<td>FT</td>
<td>PD12</td>
<td>Ta</td>
<td>PE12</td>
<td>Ta</td>
</tr>
<tr>
<td>PA13</td>
<td>FT</td>
<td>PB13</td>
<td>Ta</td>
<td>PC13</td>
<td>Ta</td>
<td>PD13</td>
<td>Ta</td>
<td>PE13</td>
<td>Ta</td>
</tr>
<tr>
<td>PA14</td>
<td>FTf</td>
<td>PB14</td>
<td>Ta</td>
<td>PC14</td>
<td>Ta</td>
<td>PD14</td>
<td>Ta</td>
<td>PE14</td>
<td>Ta</td>
</tr>
<tr>
<td>PA15</td>
<td>FTf</td>
<td>PB15</td>
<td>Ta</td>
<td>PC15</td>
<td>Ta</td>
<td>PD15</td>
<td>Ta</td>
<td>PE15</td>
<td>Ta</td>
</tr>
</tbody>
</table>

### Notes

Unless otherwise specified by a note, all I/Os are set as floating inputs during and after reset.

### Definitions

- **FT**: 5 V tolerant I/O
- **FTf**: 5 V tolerant I/O, FM+ capable
- **TTa**: 3.3 V tolerant I/O directly connected to ADC
- **TC**: Standard 3.3V I/O
- **B**: Dedicated BOOT0 pin
- **RST**: Bidirectional reset pin with embedded weak pull-up resistor

**Fast mode**: up to 400 kHz  
**Fast mode plus**: up to 1 MHz
### STM32F3 Absolute maximum ratings

#### Voltage Characteristics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Ratings</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_{DD}-V_{SS} )</td>
<td>External main supply voltage (including ( V_{DDA}, V_{BAT} ) and ( V_{DD} ))</td>
<td>-0.3</td>
<td>4.0</td>
<td>( V )</td>
</tr>
<tr>
<td>( V_{DD}-V_{DDA} )</td>
<td>Allowed voltage difference for ( V_{DD} &gt; V_{DDA} )</td>
<td></td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>( V_{IN} )</td>
<td>Input voltage on FT and FTf pins</td>
<td>( V_{SS} - 0.3 )</td>
<td>( V_{DD} + 4.0 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Input voltage on TTa pins</td>
<td>( V_{SS} - 0.3 )</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Input voltage on any other pin</td>
<td>( V_{SS} - 0.3 )</td>
<td>4.0</td>
<td></td>
</tr>
</tbody>
</table>

**STM32F3DISCOVERY BOARD:**
- VSS: 0 volts
- VDD: 3 volts
## STM32F3 Absolute maximum ratings

### Current characteristics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Ratings</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{VDD}$</td>
<td>Total current into $V_{DD}$ and VDDSDx power lines (source)$^{(2)}$</td>
<td>TBD</td>
<td></td>
</tr>
<tr>
<td>$I_{VSS}$</td>
<td>Total current out of $V_{SS}$ and VSSSD ground lines (sink)$^{(2)}$</td>
<td>TBD</td>
<td></td>
</tr>
<tr>
<td>$I_{IO(PIN)}$</td>
<td>Output current sunk by any I/O and control pin</td>
<td>25</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td>Output current source by any I/O and control pin</td>
<td>-25</td>
<td></td>
</tr>
<tr>
<td>$\Sigma I_{IO(PIN)}$</td>
<td>Total output current sunk by sum of all IOs and control pins</td>
<td>75</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td>Total output current sourced by sum of all IOs and control pins</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>$I_{INJ(PIN)}$</td>
<td>Injected current on FT, FTf and B pins$^{(3)}$</td>
<td>-5/0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Injected current on TC and RST pin$^{(4)}$</td>
<td>±5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Injected current on TTa pins$^{(5)}$</td>
<td>±5</td>
<td></td>
</tr>
<tr>
<td>$\Sigma I_{INJ(PIN)}$</td>
<td>Total injected current (sum of all I/O and control pins)$^{(6)}$</td>
<td>±25</td>
<td></td>
</tr>
</tbody>
</table>
### General input/output characteristics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_{IL} )</td>
<td>Standard I/O input low level voltage</td>
<td>-0.3</td>
<td>-</td>
<td>-</td>
<td>( 0.3V_{DD}+0.07 )</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>TTA I/O input low level voltage</td>
<td>-0.3</td>
<td>-</td>
<td>-</td>
<td>( 0.3V_{DD}+0.07 )</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>FT and FTf(^{(1)}) I/O input low level voltage</td>
<td>-0.3</td>
<td>-</td>
<td>-</td>
<td>( 0.475V_{DD}-0.2 )</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>BOOT0 input low level voltage</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>( 0.3V_{DD}-0.3 )</td>
<td>V</td>
</tr>
<tr>
<td>( V_{IH} )</td>
<td>Standard I/O input high level voltage</td>
<td>0.445(V_{DD}+0.398 )</td>
<td>-</td>
<td>-</td>
<td>( V_{DD}+0.3 )</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>TTA I/O input high level voltage</td>
<td>0.445(V_{DD}+0.398 )</td>
<td>-</td>
<td>-</td>
<td>( V_{DD}+0.3 )</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>FT and FTf(^{(1)}) I/O input high level voltage</td>
<td>0.5(V_{DD}+0.2 )</td>
<td>-</td>
<td>-</td>
<td>5.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BOOT0 input high level voltage</td>
<td>0.2(V_{DD}+0.95 )</td>
<td>-</td>
<td>-</td>
<td>5.5</td>
<td></td>
</tr>
</tbody>
</table>
# I/O port characteristics

## Output voltage characteristics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Conditions</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{OL}^{(1)}$</td>
<td>Output low level voltage for an I/O pin when 8 pins are sunk at same time</td>
<td>$I_{IO} = +8 mA$ [2.7 \text{ V} &lt; V_{DD} &lt; 3.6 \text{ V}]</td>
<td>-</td>
<td>0.4</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OH}^{(3)}$</td>
<td>Output high level voltage for an I/O pin when 8 pins are sourced at same time</td>
<td>$V_{DD} - 0.4$</td>
<td>-</td>
<td>-</td>
<td>V</td>
</tr>
<tr>
<td>$V_{CL}^{(1)}$</td>
<td>Output low level voltage for an I/O pin when 8 pins are sunk at same time</td>
<td>$I_{IO} = +8 mA$ [2.7 \text{ V} &lt; V_{DD} &lt; 3.6 \text{ V}]</td>
<td>-</td>
<td>0.4</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OH}^{(3)}$</td>
<td>Output high level voltage for an I/O pin when 8 pins are sourced at same time</td>
<td>$V_{DD} - 0.4$</td>
<td>-</td>
<td>2.4</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OL}^{(1)(4)}$</td>
<td>Output low level voltage for an I/O pin when 8 pins are sunk at same time</td>
<td>$I_{IO} = +20 mA$ [2.7 \text{ V} &lt; V_{DD} &lt; 3.6 \text{ V}]</td>
<td>-</td>
<td>1.3</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OH}^{(3)(4)}$</td>
<td>Output high level voltage for an I/O pin when 8 pins are sourced at same time</td>
<td>$V_{DD} - 1.3$</td>
<td>-</td>
<td>-</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OL}^{(1)(4)}$</td>
<td>Output low level voltage for an I/O pin when 8 pins are sunk at same time</td>
<td>$I_{IO} = +6 mA$ [2 \text{ V} &lt; V_{DD} &lt; 2.7 \text{ V}]</td>
<td>-</td>
<td>0.4</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OH}^{(3)(4)}$</td>
<td>Output high level voltage for an I/O pin when 8 pins are sourced at same time</td>
<td>$V_{DD} - 0.4$</td>
<td>-</td>
<td>-</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OLFM+}$</td>
<td>Output low level voltage for an FTf I/O pin in FM+ mode</td>
<td>$I_{IO} = +20 mA$ [2.7 \text{ V} &lt; V_{DD} &lt; 3.6 \text{ V}]</td>
<td>-</td>
<td>0.4</td>
<td>V</td>
</tr>
</tbody>
</table>
I/O Device Categories

• Input devices
  – Sensors, User-input
• Output devices
  – Actuators, Displays
• Complex I/O devices (printers, faxes, coprocessors, etc.)

Analog I/O issues
  – Voltage levels
  – Current draw
  – Sampling frequency
  – Noise

Digital I/O issues
  - Voltage levels
  - Synchronization
  - Throughput
  - Noise
Input Examples

• Sensors
  – light
  – force
  – sound
  – position
  – orientation
  – proximity
  – tactile
  – temperature
  – pressure
  – humidity
  – speed
  – acceleration
  – displacement

• User input
  – keyboards
  – joysticks
  – mouse
  – keypad
  – switches
  – touchpad
  – dial
  – slider
Output Examples

• Actuators
  – motors
  – solenoids
  – relays
  – heaters
  – lights
  – piezoelectric materials (buzzers, linear actuator)
  – speakers

• Displays
  – LED displays
  – LCD displays
  – CRT displays
  – indicator lights
  – indicator gauges
Interfacing with LED Devices

- The figure below suggests three methods for interfacing with LEDs.
- Circuit (a) and (b) are recommended for LEDs that need only small current to light.
- Circuit (c) is recommended for LEDs that need larger current to light.

An LED connected to a CMOS inverter through a current-limiting resistor

$V_{OL, min} = 0.15V$
### LED

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Condition</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luminous Intensity</td>
<td>$I_v$</td>
<td>$I_f=20mA$</td>
<td>20</td>
<td>45</td>
<td></td>
<td>mcd</td>
</tr>
<tr>
<td>Forward Voltage</td>
<td>$V_f$</td>
<td>$I_f=20mA$</td>
<td></td>
<td>1.8</td>
<td>2.2</td>
<td>V</td>
</tr>
<tr>
<td>Peak Wavelength</td>
<td>$\lambda P$</td>
<td>$I_f=20mA$</td>
<td></td>
<td>660</td>
<td></td>
<td>nm</td>
</tr>
</tbody>
</table>


### 74HC04

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Conditions</th>
<th>$25 , ^\circ C$</th>
<th>$-40 , ^\circ C \text{ to } +85 , ^\circ C$</th>
<th>$-40 , ^\circ C \text{ to } +125 , ^\circ C$</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{OL}$</td>
<td>LOW-level output voltage</td>
<td>$V_I = V_{IH} \text{ or } V_{IL}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_O = 20 \mu A; V_{CC} = 2.0 , V$</td>
<td>-</td>
<td>0</td>
<td>0.1</td>
<td>-</td>
<td>0.1</td>
<td>-</td>
</tr>
<tr>
<td>$I_O = 20 \mu A; V_{CC} = 4.5 , V$</td>
<td>-</td>
<td>0</td>
<td>0.1</td>
<td>-</td>
<td>0.1</td>
<td>-</td>
</tr>
<tr>
<td>$I_O = 20 \mu A; V_{CC} = 6.0 , V$</td>
<td>-</td>
<td>0</td>
<td>0.1</td>
<td>-</td>
<td>0.1</td>
<td>-</td>
</tr>
<tr>
<td>$I_O = 4.0 , mA; V_{CC} = 4.5 , V$</td>
<td>-</td>
<td>0.15</td>
<td>0.26</td>
<td>-</td>
<td>0.33</td>
<td>-</td>
</tr>
<tr>
<td>$I_O = 5.2 , mA; V_{CC} = 6.0 , V$</td>
<td>-</td>
<td>0.16</td>
<td>0.26</td>
<td>-</td>
<td>0.33</td>
<td>-</td>
</tr>
</tbody>
</table>

Example: Use PC[7:0] to drive eight LEDs using the circuit shown in the Figure below. Light each LED for half a second in turn and repeat assuming the GPIOC has a 72-MHz clock.

To turn on one LED at a time for half a second in turn, one should output the value $80, $40, $20, $10, $08,$04,$02, and $01 and stay for half a second in each value.
The C language version of the program is as follows:

```c
#include "stm32f30x.h"

void delaybyms(unsigned int j);

int main(void) {
    unsigned char led_tab[] = {0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01,
                                0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};

    char i=0;

    RCC->AHBENR |= RCC_AHBENR_GPIOCEN; // Enable GPIOC clock

    // PC[7:0] configuration
    RCC->AHBENR |= RCC_AHBENR_GPIOCEN; // Enable GPIOC clock
    GPIOC->MODER = GPIOC->MODER & 0xFFFF0000 | 0x00005555; // 0b01: Output
    GPIOC->OTYPER = GPIOC->OTYPER & 0xFFFFFF00; // 0b0: PP (R)
    GPIOC->OSPEEDR = GPIOC->OSPEEDR & 0xFFFF0000 | 0x0000FFFF; // 0b11: 50MHz
    GPIOC->PUPDR = GPIOC->PUPDR & 0xFFFF0000; // 0b00: no PU/PD (R)

    while (1) {
        for (i = 0; i < 16; i++) {
            GPIOC->ODR = GPIOC->ODR & 0xFFFFFF00 | led_tab[i];
            msdelay(500);
        }
    }
}

void delaybyms(unsigned int j) {
    unsigned int k, l;
    for (k=0; k<j; k++)
        for (l=0; l<1427; l++)
            gpiotest.c
```
STM32F3DISCOVERY BOARD: SCHEMATIC DETAIL

Driving a Single Seven-Segment Display

- A common cathode seven-segment display is driven by the 74HC244 via resistors.
- The output high voltage of the 74HC244 is close to 3V with a 3V power supply.
- The segment patterns for 0 to 9 are shown in the table below.

<table>
<thead>
<tr>
<th>BCD digit</th>
<th>Segments</th>
<th>Corresponding Hex Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 1 1 1 1 1 1 0</td>
<td>$7E$</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1 0 0 0 0</td>
<td>$30$</td>
</tr>
<tr>
<td>2</td>
<td>1 1 0 1 1 0 1</td>
<td>$6D$</td>
</tr>
<tr>
<td>3</td>
<td>1 1 1 1 0 0 1</td>
<td>$79$</td>
</tr>
<tr>
<td>4</td>
<td>0 1 1 0 0 1 1</td>
<td>$33$</td>
</tr>
<tr>
<td>5</td>
<td>1 0 1 1 0 1 1</td>
<td>$5B$</td>
</tr>
<tr>
<td>6</td>
<td>1 0 1 1 1 1 1</td>
<td>$5F$</td>
</tr>
<tr>
<td>7</td>
<td>1 1 1 0 0 0 0</td>
<td>$70$</td>
</tr>
<tr>
<td>8</td>
<td>1 1 1 1 1 1 1</td>
<td>$7F$</td>
</tr>
<tr>
<td>9</td>
<td>1 1 1 1 0 1 1</td>
<td>$7B$</td>
</tr>
</tbody>
</table>

Table: Specifications of TDSL1150 and TDSL1160 parts

<table>
<thead>
<tr>
<th>PART</th>
<th>COLOR</th>
<th>LUMINOUS INTENSITY (μcd)</th>
<th>$I_F$ (mA)</th>
<th>WAVELENGTH (nm)</th>
<th>$I_F$ (mA)</th>
<th>FORWARD VOLTAGE (V)</th>
<th>$I_F$ (mA)</th>
<th>CIRCUITRY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MIN. TYP. MAX.</td>
<td>MIN. TYP. MAX.</td>
<td>MIN. TYP. MAX.</td>
<td>MIN. TYP. MAX.</td>
<td>MIN. TYP. MAX.</td>
<td>MIN. TYP. MAX.</td>
<td></td>
</tr>
<tr>
<td>TDSL1150</td>
<td>Red</td>
<td>180 260 - 2</td>
<td>612 - 625 2</td>
<td>- 1.8 2.4</td>
<td>2</td>
<td>Common anode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDSL1160</td>
<td>Red</td>
<td>180 260 - 2</td>
<td>612 - 625 2</td>
<td>- 1.8 2.4</td>
<td>2</td>
<td>Common cathode</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Driving Multiple Seven-Segment Displays

- Time multiplexing technique is often used to drive multiple displays in order to save I/O pins.
- One parallel port is used to drive the segment pattern and the other port turns on one display at a time. Each display is turned on and then off many times within a second. The persistence of vision make us feel that all displays are turned on simultaneously.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Test Conditions</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{CE(sat)}$</td>
<td>Collector-emitter Saturation Voltage</td>
<td>$I_C = 150$ mA, $I_E = 15$ mA</td>
<td>0.4</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$I_C = 500$ mA, $I_E = 50$ mA</td>
<td>1.8</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{BE(sat)}$</td>
<td>Base-emitter Saturation Voltage</td>
<td>$I_B = 150$ mA, $I_E = 15$ mA</td>
<td>1.3</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$I_B = 500$ mA, $I_E = 50$ mA</td>
<td>2.8</td>
<td>V</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Port C and Port D together drive six seven-segment displays.*
**Example:** Write a sequence of instructions to display 4 on the seven-segment display #4 in the figure above.

**Solution:** To display the digit 4 on the display #4, we need to:

- Output the hex value $33$ to port C
- Set the PD4 pin to 1
- Clear pins PD5 and PD3...PD0 to 0

In C language:

```c
// Configure PC[6:0] & PD[5:0] as GP output + PP
GPIOC->ODR = GPIOD->ODR & 0xFFFFFFFF80 | 0x33;
GPIOD->ODR = GPIOD->ODR & 0xFFFFFFFFFC0 | 0x10;
```
Example: Write a program to display 123456 on the six seven-segment displays shown in the figure below.

Solution: Display 123456 on display #5, #4, #3, #2, #1, and #0, respectively.

• The values to be output to Port C and Port D to display one digit at a time is shown in the table below.

<table>
<thead>
<tr>
<th>seven-segment display</th>
<th>displayed BCD digit</th>
<th>Port B</th>
<th>Port K</th>
</tr>
</thead>
<tbody>
<tr>
<td>#5</td>
<td>1</td>
<td>$30</td>
<td>$20</td>
</tr>
<tr>
<td>#4</td>
<td>2</td>
<td>$6D</td>
<td>$10</td>
</tr>
<tr>
<td>#3</td>
<td>3</td>
<td>$79</td>
<td>$08</td>
</tr>
<tr>
<td>#2</td>
<td>4</td>
<td>$33</td>
<td>$04</td>
</tr>
<tr>
<td>#1</td>
<td>5</td>
<td>$5B</td>
<td>$02</td>
</tr>
<tr>
<td>#0</td>
<td>6</td>
<td>$5F</td>
<td>$01</td>
</tr>
</tbody>
</table>

Table of display patterns for this example
Time-multiplexed seven-segment display algorithm

1. Start
2. Set X to the address of the display table
3. Output the byte at [X] to port C
4. Output the byte at [X]+1 to Port D
5. Increment X by 2
6. Wait for 1 ms
7. If X = display + 12, go to step 8. Otherwise, go to step 2.

8. Yes? If yes, stop. If no, go to step 2.
int main (void)
{
    char disp_tab[6][2] = {{0x30,0x20},{0x6D,0x10},{0x79,0x08},
                          {0x33,0x04},{0x5B,0x02},{0x5F,0x01}};

    char i;
    // configure PC[6:0] for output
    // configure PD[5:0] for output
    while (1) {
        for (i = 0; i < 6; i++) {
            // output the segment pattern
            GPIOC->ODR = GPIOC->ODR & 0xFFFFFF80 | disp_tab[i][0];
            // turn on the display
            GPIOD->ODR = GPIOD->ODR & 0xFFFFFFFC0 | disp_tab[i][1];
            delaybyms(1);
        }
    }
}
Stepper Motor Control (1 of 7)

- It is digital in nature and provides high degree of control.
- In its simplest form, a stepper motor has a permanent magnet rotor and a stator consisting of two coils. The rotor aligns with the stator coil that is energized.
- By changing the coil that is energized, the rotor is turned.
- Next four figures illustrate how the rotor rotates clockwise in full step.

http://homepage.cem.itesm.mx/carbajal/Microcontrollers/ASSIGNMENTS/labs/pas.swf
Stepper Motor Control (2 of 7)

Stepper motor clockwise rotation in full step (1 of 2)

Stepper motor full step 1

Stepper motor full step 2

Stepper motor clockwise rotation in full step (1 of 2)
Stepper Motor Control (3 of 7)

Stepper motor full step 3

Stepper motor full step 4

Stepper motor clockwise rotation in full step (2 of 2)
Stepper Motor Control (4 of 7)

• Next figure illustrates how the rotor rotates counter-clockwise in full step.

Full-step counter-clockwise operation of step motor
In a four-pole stepper motor shown before, a full step is 90 degrees.

The stepper motor may also operate with half step. A half step occurs when the rotor (in a four-pole step) is moved to eight discrete positions (45°).

To operate the stepper motor in half steps, sometimes both coils may have to be on at the same time. When two coils in close proximity are energized, there is a resultant magnetic field whose center will depend on the relative strengths of the two magnetic fields.

The next figure illustrates the half-stepping sequence.
Stepper Motor Control (6 of 7)

Half-step operation of the stepper motor
Stepper Motor Control (7 of 7)

The actual stator of a real motor has more segments than previously indicated. One example is shown in the next figure. The step sizes of the stepper motors may vary from approximately 0.72° to 90°. The most common step sizes are 1.8°, 7.5°, and 15°.
Stepper Motor Drivers (1 of 4)

- Driving a step motor involves applying a series of voltages to the coils of the motor.
- A subset of coils is energized at a time to cause the motor to rotate one step. The pattern of coils energized must be followed exactly for the motor to work correctly.
- A microcontroller can easily time the duration that the coil is energized, and control the speed of the stepper motor in a precise manner.
- The circuit in the figure below shows how the transistors are used to switch the current to each of the four coils of the stepper motor.
- The diodes in the figure below are called *fly back diodes* and are used to protect the transistors from reverse bias.
- The transistor loads are the windings in the stepper motor. The windings are inductors, storing energy in a magnetic field.
- When the current is cut off, the inductor dispenses its stored energy in the form of an electric current.
- This current attempts to flow through the transistor, reversely biasing its collector-emitter pair. The diodes are placed to prevent this current from going through the transistors.
Stepper Motor Drivers (2 of 4)

Driving the stepper motor
Stepper Motor Drivers (3 of 4)

- The normal full-step sequence should be used for high-torque applications.
- For lower-torque applications the half-step mode is used.
- The microcontroller outputs the voltage pattern in the sequence shown in these tables.
- The tables are circular. The values may be output in the order as shown in the table, which will rotate the motor clockwise; or in the reverse order, which will rotate the motor counterclockwise.
- A delay of about 5 to 15ms is required between two steps to prevent motor from missing steps.

### Full-step sequence for clockwise rotation

<table>
<thead>
<tr>
<th>Step</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>1010</td>
</tr>
<tr>
<td>2</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>1001</td>
</tr>
<tr>
<td>3</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>0101</td>
</tr>
<tr>
<td>4</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>0110</td>
</tr>
<tr>
<td>1</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>1010</td>
</tr>
</tbody>
</table>

### Half-step sequence for clockwise rotation

<table>
<thead>
<tr>
<th>Step</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>1010</td>
</tr>
<tr>
<td>2</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>1000</td>
</tr>
<tr>
<td>3</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>1001</td>
</tr>
<tr>
<td>4</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>0001</td>
</tr>
<tr>
<td>5</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>0100</td>
</tr>
<tr>
<td>7</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>0110</td>
</tr>
<tr>
<td>8</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>0010</td>
</tr>
<tr>
<td>1</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>off</td>
<td>1010</td>
</tr>
</tbody>
</table>
**Example:** Assuming that pins PP3...PP0 are used to drive the four transistor in the figure above, write a subroutine to rotate the stepper motor clockwise one cycle using the half-step sequence.

```c
#include <stm32f30x.h>

const unsigned char sequence[] = {0x05, 0x01, 0x09, 0x08, 0x0A, 0x02, 0x04, 0x06};

int main( void ) {
    unsigned int i,j;

    // Configure GPIOC[3:0] as output, PP, high speed, no pullup/pulldown
    RCC->AHBENR    |= 0x00080000; // Activating GPIOC clock source
    GPIOC->MODER   |= 0x00000055; // Configuring GPIOC[3:0] as outputs
    GPIOC->OSPEEDR &= 0xFFFFFFAA; // Selecting low-speed on GPIOC[3:0]
    while (1) {
        for (i=0;i<8;i++) {
            GPIOC->ODR = sequence[i]; // Sequence number output
            for ( j=0;j<10000;j++ ); // Delay loop
        }
    }
}
Interfacing with DIP Switches (1 of 2)

- Switches are often grouped together. It is most common to have four or eight switches in a DIP package.
- DIP switches are often used to provide setup information to the microcontroller. After power is turned on, the microcontroller reads the settings of the DIP switches and performs accordingly.

Connecting a set of eight DIP switches to port D of the MCU

STM32F3
\[ R_{\text{PULL-UP}} = R_{\text{PULL-DOWN}} = 40\,\Omega \]
Interfacing with DIP Switches (2 of 2)

• **Example** Write a sequence of instructions to read the value from an eight-switch DIP connected to GPIOD of the STM32 into accumulator A.

• **Solution**

In C language

```c
void main () {
    char xx;
    RCC->AHBENR |= 1 << 20;       // Enable GPIOD clock
    GPIOD->MODER &= 0xFFFF0000;   // PD[7:0] as input (reset value)
    GPIOD->PUPDR |= 0x00005555;   // Pull-up resistors
    xx = GPIOD->IDR & 0x000000FF; // clean value
}
```
Hardware Debouncing Techniques

- SR latches
- Non-inverting CMOS gates
- Integrating debouncer

Figure 7.42 Hardware debouncing techniques
Software Debouncing Technique

• The most popular and simple one has been the **wait and see** method.
  – In this method, the program simply waits for about 10 ms and reexamines the same key again to see if it is still pressed.
Interfacing the MCU to a Keypad

- A keypad usually consists of 12 to 24 keys and is adequate for many applications.
- Like a keyboard, a keypad also needs debouncing.
- A 16-key keypad can be easily interfaced to one of the MCU parallel ports.
- A circuit that interfaces a 16-key keypad is shown in the figure below (left). Note: pins PC[7:4] each control four keys.

<table>
<thead>
<tr>
<th>PC7</th>
<th>PC6</th>
<th>PC5</th>
<th>PC4</th>
<th>Selected keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0, 1, 2, and 3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4, 5, 6, and 7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8, 9, A, and B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>C, D, E, and F</td>
</tr>
</tbody>
</table>

Sixteen-key keypad row selections

Sixteen-key keypad connected to the MCU
```c
# define keypad GPIOC->IDR  // keypad port
# define keypad_dir GPIOC->MODER  // keypad port direction register

// mask is row mask, cmask is column mask, row is the row being scanned, col is the
// column being scanned

// **************************************************************************************
char getkey (void)
{
    char rmask, cmask, row, col;
    char temp, keycode;

    keypad_dir = (keypad_dir & 0xFFFF0000) | 0x00005500; // Configure lower bits[3:0] as input, bits[7:4] as output
    while (1) {
        rmask  = 0xEF;
        for (row = 0; row < 4; row++){
            cmask  = 0x01;
            GPIOC->ODR &= rmask;  // select the current row
            for (col = 0; col < 4; col++){
                // key switch detected pressed
                if (!((keypad & cmask)){
                    delayby10ms(1);
                    if(!((keypad & cmask)){    // check the same key again
                        keycode = row * 4 + col;
                        if(keycode < 10)
                            return (0x30 + keycode);
                        else
                            return (0x37 + keycode);
                    }
                    cmask = cmask << 1;
                }
                rmask = (rmask << 1) | 0xEF;
            }
        }
    }
}

Adapted from “The HCS12/9S12: An Introduction to Software and Hardware Interfacing”, HUANG (2010)
```
Example: Write a C program to read a character from the keypad shown in the figure above. This program will perform keypad scanning, debouncing, and ASCII code lookup.

Solution:

```c
void delaybyms (unsigned int); // prototype

char get_key (void) {
    RCC->AHBENR |= 1 << 19; // Enable GPIOC clock
    GPIOC->MODER |= 0x5500; // configure PC[7:4] for output and PC[3:0] for input

    while (1) {
        GPIOC->ODR = 0xE0; // prepare to scan the row controlled by PC4

        if (!(GPIOC->IDR & 0x01)) {
            delaybyms (10);
            if (!(GPIOC->IDR & 0x01))
                return 0x30; // return ASCII code of 0
        }

        if (!(GPIOC->IDR & 0x02)) {
            delaybyms (10);
            if (!(GPIOC->IDR & 0x02))
                return 0x31; // return ASCII code of 1
        }
    }
}
```
if (!(GPIOC->IDR & 0x04)) {
    delaybyms (10);
    if (!(PORTC & 0x04))
        return 0x32; // return ASCII code of 2
}
if (!(GPIOC->IDR & 0x08)) {
    delaybyms (10);
    if (!(GPIOC->IDR & 0x08))
        return 0x33; // return ASCII code of 3
}

GPIOC->ODR = 0xD0; // set PC5 to low to scan second row

if (!(GPIOC->IDR & 0x01)) {
    delayby0ms (10);
    if (!(GPIOC->IDR & 0x01))
        return 0x34; // return ASCII code of 4
}
if (!(GPIOC->IDR & 0x02)) {
    delaybyms (10);
    if (!(GPIOC->IDR & 0x02))
        return 0x35; // return ASCII code of 5
}
if (!(GPIOC->IDR & 0x04)) {
    delaybyms (10);
    if (!(GPIOC->IDR & 0x04))
        return 0x36; // return ASCII code of 6
}
if (!(GPIOC->IDR & 0x08)) {
    delaybyms (10);
    if (!(GPIOC->IDR & 0x08))
        return 0x37; // return ASCII code of 7
}
GPIOC->ODR = 0xB0; // set PC6 to low to scan the third row
if (!(GPIOC->IDR & 0x01)) {
    delaybyms (10);
    if (!(GPIOC->IDR & 0x01))
        return 0x38; // return ASCII code of 8
}
if (!(GPIOC->IDR & 0x02)) {
    delaybyms (10);
    if (!(GPIOC->IDR & 0x02))
        return 0x39; // return ASCII code of 8
}
if (!(GPIOC->IDR & 0x04)) {
    wait_10ms ( );
    if (!(GPIOC->IDR & 0x04))
        return 0x41; // return ASCII code of A
}
if (!(GPIOC->IDR & 0x08)) {
    delaybyms (10);
    if (!(GPIOC->IDR & 0x08))
        return 0x42; // return ASCII code of B
}

GPIOC->ODR = 0x70; // set PC7 to low to scan the fourth row

if (!(GPIOC->IDR & 0x01)) {
    delaybyms (10);
    if (!(GPIOC->IDR & 0x01))
        return 0x43; // return ASCII code of C
}
if (!(GPIOC->IDR & 0x02)) {
    delaybyms (10);
    if (!(GPIOC->IDR & 0x02))
        return 0x44; // return ASCII code of D
}
if (!(GPIOC->IDR & 0x04)) {
    wait_10ms ( );
    if (!(GPIOC->IDR & 0x04))
        return 0x45; // return ASCII code of E
}
if (!(GPIOC->IDR & 0x08)) {
    delaybyms (10);
    if (!(GPIOC->IDR & 0x08))
        return 0x46; // return ASCII code of F
}
Debouncing User Switch

![Circuit Diagram](image)
Liquid Crystal Display (LCD)

- The most common type of LCD allows the light to pass through when activated.
- An LCD segment is activated when a low frequency bipolar signal in the range of 30 Hz to 1KHz is applied to it.
- LCD can display characters and graphics.
- LCDs are often sold in a module with LCDs and controller unit built in.
- The Hitachi HD44780 is the most popular LCD controller being used today.

Basic construction of an LCD

https://www.youtube.com/watch?v=O3aITfU_UvE
A HD44780-Based LCD Kit

- Display capability: 4 x 20
- Uses the HD44780 as the controller as shown in the figure below.
- Pins DB7~DB0 (data port) are used to exchange data with the CPU.
- E input should be connected to one of the address decoder output or I/O pin.
- The RS signal selects instruction register (0) or data register (1).
- The VEE signal allows the user to adjust the LCD contrast.
- The HD44780 can be configured to display 1-line, 2-line, and 4-line information.
### HD44780 Commands (1 of 3)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Description</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear display</td>
<td>0 0 0 0 0 0 1</td>
<td>Clears display and returns cursor to the home position (address 0).</td>
<td>1.64 ms</td>
</tr>
<tr>
<td>Cursor home</td>
<td>0 0 0 0 0 0 1</td>
<td>Returns cursor to home position (address 0). Also returns display being shifted to the original position. DDRAM contents remain unchanged.</td>
<td>1.64 ms</td>
</tr>
<tr>
<td>Entry mode set</td>
<td>0 0 0 0 0 0 0 1 I/D S</td>
<td>Set cursor move direction (I/D), specifies to shift the display (S). These operations are performed during data read/write.</td>
<td>40 μs</td>
</tr>
<tr>
<td>Display on/off control</td>
<td>0 0 0 0 0 0 1 D C B</td>
<td>Sets on/off of all display (D), cursor on/off (C) and blink of cursor position character (B).</td>
<td>40 μs</td>
</tr>
<tr>
<td>Cursor/display shift</td>
<td>0 0 0 0 0 1 S/C R/L *</td>
<td>Sets cursor-move or display- (S/C), shift direction (R/L). DDRAM contents remains unchanged.</td>
<td>40 μs</td>
</tr>
<tr>
<td>Function set</td>
<td>0 0 0 0 1 D L N F *</td>
<td>Sets interface data length (DL), number of display line (N) and character font (F).</td>
<td>40 μs</td>
</tr>
<tr>
<td>Set CGRAM address</td>
<td>0 0 0 1 CGRAM address</td>
<td>Sets the CGRAM address. CGRAM data is sent and received after this setting.</td>
<td>40 μs</td>
</tr>
<tr>
<td>Set DDRAM address</td>
<td>0 0 1 DDRAM address</td>
<td>Sets the DDRAM address. DDRAM data is sent and received after this setting.</td>
<td>40 μs</td>
</tr>
<tr>
<td>Read busy flag and address counter</td>
<td>0 1 B F CGRAM/DDRAM address</td>
<td>Reads busy flag (BF) indicating internal operation is being performed and reads CGRAM or DDRAM address counter contents (depending on previous instruction).</td>
<td>0 μs</td>
</tr>
<tr>
<td>Write CGRAM or DDRAM</td>
<td>1 0 write data</td>
<td>Writes data to CGRAM or DDRAM.</td>
<td>40 μs</td>
</tr>
<tr>
<td>Read from CGRAM or DDRAM</td>
<td>1 1 read data</td>
<td>Reads data from CGRAM or DDRAM.</td>
<td>40 μs</td>
</tr>
</tbody>
</table>

**HD4478U instruction set**
### HD44780 Commands (2 of 3)

<table>
<thead>
<tr>
<th>Bit name</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/D</td>
<td>0 = decrement cursor position. 1 = increment cursor position</td>
</tr>
<tr>
<td>S</td>
<td>0 = no display shift. 1 = display shift</td>
</tr>
<tr>
<td>D</td>
<td>0 = display off 1 = display on</td>
</tr>
<tr>
<td>C</td>
<td>0 = cursor off 1 = cursor on</td>
</tr>
<tr>
<td>B</td>
<td>0 = cursor blink off 1 = cursor blink on</td>
</tr>
<tr>
<td>S/C</td>
<td>0 = move cursor 1 = shift display</td>
</tr>
<tr>
<td>R/L</td>
<td>0 = shift left 1 = shift right</td>
</tr>
<tr>
<td>DL</td>
<td>0 = 4-bit interface 1 = 8-bit interface</td>
</tr>
<tr>
<td>N</td>
<td>0 = 1/8 or 1/11 duty (1 line) 1 = 1/16 duty (2 lines)</td>
</tr>
<tr>
<td>F</td>
<td>0 = 5x8 dots 1 = 5 x 10 dots</td>
</tr>
<tr>
<td>BF</td>
<td>0 = can accept instruction 1 = internal operation in progress</td>
</tr>
</tbody>
</table>

LCD instruction bit names
HD44780 Commands (3 of 3)

- The HD44780 has a display data RAM (DDRAM) to store data to be displayed on the LCD.

### DDRAM address usage for a 2-line LCD

<table>
<thead>
<tr>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>0A</th>
<th>0B</th>
<th>0C</th>
<th>0D</th>
<th>0E</th>
<th>0F</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>4A</td>
<td>4B</td>
<td>4C</td>
<td>4D</td>
<td>4E</td>
<td>4F</td>
</tr>
</tbody>
</table>

### DDRAM address usage for a 1-line LCD

<table>
<thead>
<tr>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>40</th>
<th>41</th>
<th>42</th>
<th>43</th>
<th>44</th>
<th>45</th>
<th>46</th>
<th>47</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCDEFGHIJKLMNOP</td>
<td>abcdefghijklmnop</td>
<td>ABCDEFGH01234567</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Interfacing the HD44780 with the MCU

- One can treat the LCD kit as an I/O device and use an I/O port and several other I/O pins as control signals.
- The interface can be 4 bits or 8 bits.
- To read or write the LCD successfully, one must satisfy the timing requirements of the LCD. The timing diagrams for read and write are shown in the next two figures.

![LCD interface example (8-bit bus)](image1)

![LCD interface example (4-bit bus)](image2)
HD4478U LCD controller read timing diagram

HD4478U LCD controller write timing diagram
### HDD4478U bus timing parameters (2 MHz operation)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Min</th>
<th>Typ</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_CYCLE</td>
<td>Enable cycle time</td>
<td>500</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>PW_EH</td>
<td>Enable pulse width (high level)</td>
<td>230</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>t_Er, t_Ef</td>
<td>Enable rise and decay time</td>
<td></td>
<td>-</td>
<td>20</td>
<td>ns</td>
</tr>
<tr>
<td>t_AB</td>
<td>Address setup time, RS, R/W, E</td>
<td>40</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>t_DDR</td>
<td>Data delay time</td>
<td></td>
<td>-</td>
<td>160</td>
<td>ns</td>
</tr>
<tr>
<td>t_DSW</td>
<td>Data setup time</td>
<td>80</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>t_H</td>
<td>Data hold time (write)</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>t_DHR</td>
<td>Data hold time (read)</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>t_AH</td>
<td>Address hold time</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
</tbody>
</table>
• Write a function to send a command to the LCD kit
  – Most LCD commands are completed in 40 ms.
  – If the function waits for 40 ms after performing the specified operation, then most commands will be completed when the function returns.

```c
#define lcdPort   GPIOD->ODR    // Port D drives LCD data pins
#define lcdE      0x40           // E signal (PC6)
#define lcdRW     0x20           // R/W signal (PC5) Connected to ground
#define lcdRS     0x10           // RS signal (PC4)
#define lcdCtl    GPIOC->ODR    // LCD control port direction
```
void cmd2lcd (char cmd) {
    char temp;
    char xa, xb;
    lcdCtl &= ~(lcdRS+lcdRW); // select instruction register & pull R/W low
    lcdCtl |= lcdE; // pull E signal to high
    lcdPort = cmd; // output command
    xa = 1; // dummy statements to lengthen E
    xb = 2; //       "
    lcdCtl &= ~lcdE; // pull E signal to low
    lcdCtl |= lcdRW; // pull R/W to high
    delayby50us(1); // wait until the command is complete
}

Send Command to LCD

HD4478U LCD controller write timing diagram
```c
void putc2lcd(char cx) {
    char temp;
    char xa, xb;
    lcdCtl |= lcdRS; // select LCD data register and pull R/W high
    lcdCtl &= ~lcdRW; // pull R/W to low
    lcdCtl |= lcdE;   // pull E signal to high
    lcdPort = cx;     // output data byte
    xa = 1;           // create enough width for E
    xb = 2;           // create enough width for E
    lcdCtl &= ~lcdE;  // pull E to low
    lcdCtl |= lcdRW;  // pull R/W signal to high
    delayby50us(1);
}
```

**Send data to LCD**

HD4478U LCD controller write timing diagram
Data String to LCD

• Function to output a string terminated by a NULL character

```c
void puts2lcd (char *ptr) {
    while (*ptr) {
        putc2lcd(*ptr);
        ptr++;
    }
}
```
8-bit Initialization

Power on

Wait for more than 15 ms after $V_{CC}$ rises to 4.5 V

RS RW DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0 0 0 0 1 1 * * *

Wait for more than 4.1 ms

RS RW DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0 0 0 0 1 1 * * *

Wait for more than 100 μs

RS RW DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0 0 0 0 1 1 * * *

Initialization ends

Function set (Interface is 8 bits long.)

BF cannot be checked before this instruction.

BF can be checked after the following instructions. When BF is not checked, the waiting time between instructions is longer than the execution instruction time. (See Table 6.)

Function set (Interface is 8 bits long. Specify the number of display lines and character font.)

The number of display lines and character font cannot be changed after this point.

Display off

Display clear

Entry mode set
The function to configure LCD sends four commands to the LCD kit

- Entry mode set
- Display on/off
- Function set
- Clear display

```c
void initlcd(void) {
    // configure lcdPort port (GPIOD) as output
    GPIOD->MODER |= 0x00005555;
    // configure LCD control pins (PC6, PC5, & PC4) as outputs
    GPIOC->MODER |= 0x00001500;
    delayby100ms(5); // wait for LCD to become ready
    cmd2lcd (0x38);  // set 8-bit data, 2-line display, 5x8 font
    cmd2lcd (0x0F);  // turn on display, cursor, blinking
    cmd2lcd (0x06);  // move cursor right
    cmd2lcd (0x01);  // clear screen, move cursor to home
    delayby1ms (2);  // wait until "clear display" command is complete
}
```
**Example** Write an C program to test the previous four subroutines by displaying the following messages on two lines:

```
  hello world!
  I am ready!
```

```c
int main (void) {
  char *msg1 = "hello world!";
  char *msg2 = "I am ready!";
  init1cd();
  cmd2lcd(0x80); // move cursor to the 1st column of row 1
  puts2lcd(msg1);
  cmd2lcd(0xC0); // move cursor to 2nd row, 1st column
  puts2lcd(msg2);
}
```
4-bit Initialization

1. Power on
   - Wait for more than 15 ms after V_{CC} rises to 4.5 V
   - RS, RW, DB7, DB6, DB5, DB4: 0 0 0 0 1 1

2. Wait for more than 4.1 ms
   - RS, RW, DB7, DB6, DB5, DB4: 0 0 0 0 1 1

3. Wait for more than 100 μs
   - RS, RW, DB7, DB6, DB5, DB4: 0 0 0 0 1 1

4. Function set (Interface is 8 bits long)
   - BS cannot be checked before this instruction.

5. Function set (Interface is 8 bits long)
   - BS cannot be checked before this instruction.

6. Function set (Interface is 8 bits long)
   - BS cannot be checked before this instruction.

7. Function set (Set interface to be 4 bits long)
   - Interface is 8 bits in length.
   - BS can be checked after the following instructions.
   - BS cannot be checked before the waiting time between instructions is longer than the execution instruction time. (See Table 6.)

   - Function set (Interface is 4 bits long. Specify the number of display lines and character font.)
   - The number of display lines and character font cannot be changed after this point.
   - Display off
   - Display clear
   - Entry mode set

Initialization ends