

## DgTelli2c – a 4-character display I2C/SMBus - 3 to 5V hexa and decimal, plus alpha and segment mode

The **DgTelli2c** is a low power miniature 4-digit display that works from 3V (15mA) to 5.5V (30mA). Also available now with a larger and cheaper display. What's great is its Ascii mode and a direct segment mode that allows to do special graphic effects. A scrolling text is easy to program: just send the next character with the correct command.

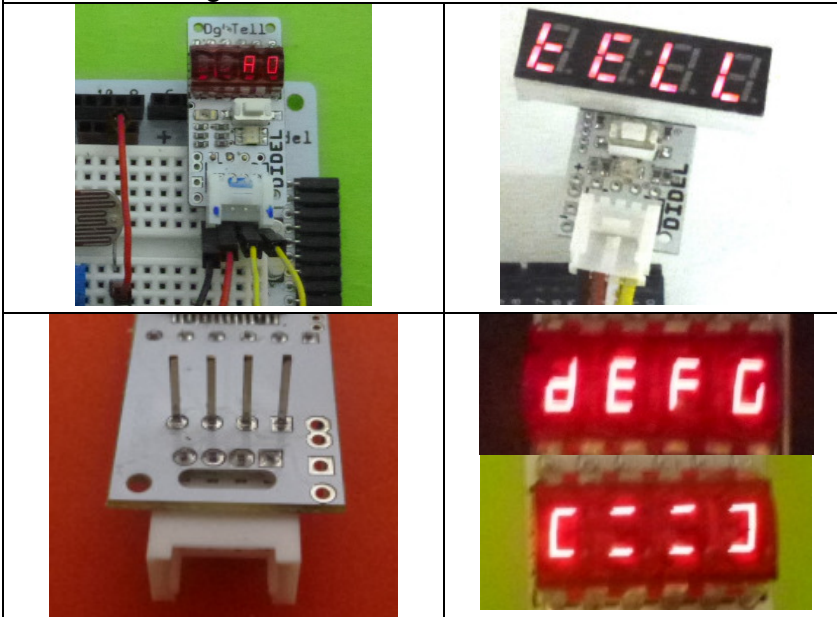
Another great feature is the possible local conversion to decimal. One thinks decimal while programming, but variables are in hexa. Default mode is decimal, but depress the button and it is converted to decimal (if possible, max 9999 = 0x270F). Five modes are provided, easy to select.

A Grove connector is installed, you can add a 2.54mm strip male or female if you prefer.

The side connector is for the DiTell 1-wire debug display and must be ignored.



Grove connector pinout:  
Gnd Vcc SDA SCL



### I2C transfer

In order to update the display, a command and 2 bytes are transmitted using any I2C library.

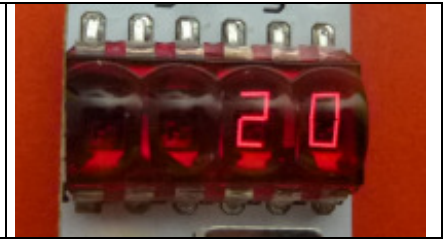
```
//TestWriteI2C.ino
#include <Wire.h> // Arduino library
#define AdTelli 0x20
void setup() { Wire.begin(); }
void loop() {
  Wire.beginTransmission(AdTelli);
  Wire.write(0x2); // command
  Wire.write(0x23);
  Wire.write(0xFC);
  Wire.endTransmission();
  for (;;) //stop
}
```



Well, this is just to show that you can write a dirty simple program to do what all other I2C display can do (and you do not need to load special software). The DgTellI2C can do a lot more.

### Select I2C address

At power-up, the 7-bit I2C address is shown for 3 seconds. Predefined address is 0x20. You can change it by depressing the push button before the display changes. A limited number of addresses are proposed, so you can go through easily and come back to 0x20. Of course, the address is saved on Eeprom.



### Commands

The DgTell has 6 commands for working with numbers or text, and there are more options in both cases. For numbers, you can decide for hexadecimal, or for the decimal equivalent, locally converted. For text, you can use the built-in character generator or use the segment mode, and use any of the 128 segment combinations, plus decimal dot.

### Write a number

As shown in the first page example, command 2 allows to transfer 16 bits that will be displayed as an hexadecimal or BCD number. Hexa is a shorthand for the 16 bits, stored in memory, displayed as 4 hexadecimal digits, 0 to F. BCD uses only 10 of these 4-bit combinations, and shows decimal digits 0 to 9;

Command **Tell** = 2 writes a 16 bit value.

Now understand the local processor can convert binary to decimal. This is what does the Serial.print (number,DEC); The binary value in memory is converted to decimal. The difference is we can display numbers to 9999 and not 65535 if the number is 0xFFFF. The DgTell binary to decimal routine will display - - - if the hex value is greater than 0x230F = 9999

How to call the conversion routine? Two possibilities:

- 1) you look at what is displayed and you prefer to see it in decimal – depress the push button. Depress again to come back.
- 2) when you write the I2C order, you know you need a decimal presentation, Command 1 will set the mode for you. The command must follow writing the number; by default you write an hex number. You can change its presentation with command 1, named Mode.

Several decimal modes, provide the flexibility to work with one 16 bits word or two 8-bit numbers. This can be preselected with command 1, or changed when you look at the display, not receiving new commands.

Using the push button, you go from one picture to the next. The programmed mode will not take care of what you may have selected on the push button.

Command **Mode** = 1

Value **Hexa** = 0 for hexadecimal. 0000 - FFFF



Value **Decimal** =4 for decimal 0000 – 9999


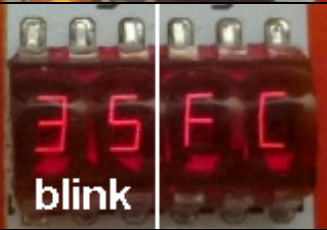
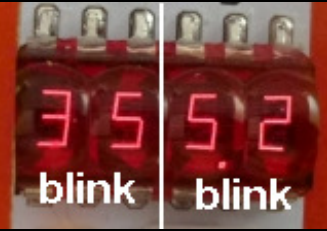
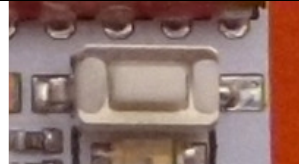
The decimal dots are on to show the difference.  
A binary value greater than 0x270F will be converted to 9999, and shown as 4 dashes -. -. -. .



Frequently, the display will be programmed to show 2 sensor values, two 8-bit numbers. It will happen the high byte should be preferably shown in hex, and the low byte in decimal. DgTell can do it, but how to make the difference?

Bytes in decimal are blinking. How to show a value up to 255? from 100 to 199, the left dot is on, from 200 to 255, it is the right dot.

FC has been converted to 252, the left dot is on.

<p>Value <b>HighDecLowHex</b> =8 First byte is converted to decimal FC has been converted to 252, the left dot is on.</p>	
<p>Value <b>HighHexLowDec</b> =16 0x23 has been converted to 35 (2*16+3).</p>	
<p>Value <b>HighDecLowDec</b> =32 0x23 --&gt; 35 , 0xFC --&gt; (2)52</p>	
<p>The push button goes through the 5 values when a number is displayed. It is not saved on Eeprom, a next command on Mode will modify it.</p>	

## Alphanumeric and segment mode

Four commands set the DgTell in text or segment (graphic) appearance.

Command **AfAB** = 3 writes the first 2 positions.

Command **AfCD** = 4 writes the next 2 positions.

These 2 commands are compatible with SMbus and Python - write\_word\_data(adr,cmd,val16) .

With the Arduino Wire library, it is possible to write 4 bytes after command 3.






































Python encourages the use of block transfers – write\_block\_data (adr,cmd,long[]) we support it

Command **TextBlock** = 5 The format is select (0x20), command (5), block length (4), data (4 bytes).

A last command facilitates scrolling texts. One sends one character at a time followed by a delay of 200 to 400 ms.

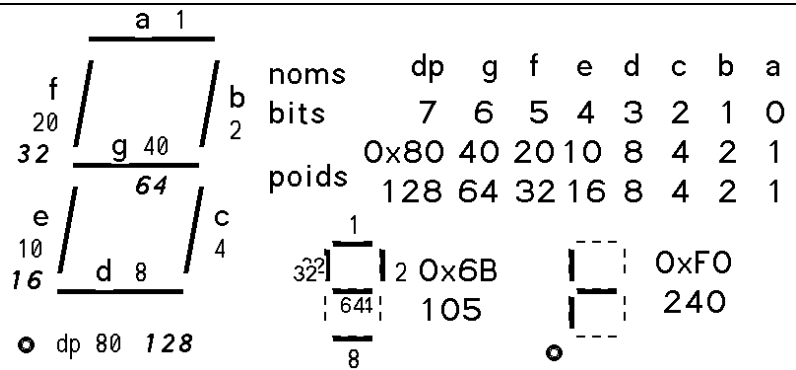
Command **Scroll** = 6 shift left and insert code.

By default, these commands call the built-in Ascii character generator.

	63	<b>0</b>	48		93	@	64		14	<b>J</b>	74		120	<b>T</b>	84
	6	<b>1</b>	49		119	<b>A</b>	65		112	<b>K</b>	75		62	<b>U</b>	85
	91	<b>2</b>	50		124	<b>B</b>	66		56	<b>L</b>	76		54	<b>V</b>	86
	79	<b>3</b>	51		88	<b>C</b>	67		85	<b>M</b>	77		106	<b>W</b>	87
	102	<b>4</b>	52		94	<b>D</b>	68		84	<b>N</b>	78		73	<b>X</b>	88
	109	<b>5</b>	53		121	<b>E</b>	69		63	<b>O</b>	79		110	<b>Y</b>	89
	125	<b>6</b>	54		113	<b>F</b>	70		115	<b>P</b>	80		27	<b>Z</b>	90
	7	<b>7</b>	55		61	<b>G</b>	71		103	<b>Q</b>	81	segments ASCII			
	127	<b>8</b>	56		118	<b>H</b>	72		80	<b>R</b>	82				
	111	<b>9</b>	57		48	<b>I</b>	73		45	<b>S</b>	83				

## Segment mode

Bytes control the segments directly. It is easy to add the weight of the segment to get the value to be send to the display.  
For instance, 57,9,9,15 (decimal) give



Segment mode is using the same command as alphanumeric commands. The Mode must be set after sending the text, in case of change of mode value.

<p>Command <b>Mode</b> = 1</p> <p>Value <b>Ascii</b> = 1 Not required after a write command</p> <p>Read J K L M</p>	
<p>Value <b>Segments</b> = 3 Segments</p> <p>ABCD in segment mode 0x41 42 43 44</p>	

You can also use the push button to switch between these two modes.

## Définition file and functions for Arduino/Diduino

<pre>#define Xtell 0x20 #define Id 0 #define SegBlock 1 #define TxtBlock 2 #define AfScroll 3 #define AfHex 4 #define AfDec 5 #define AfMode 6 #define AfAB 7 #define AfCD 8  Functions v8 = ReadId(); test 0 WriteHex (v16); test 1 WriteDec (v16); test 2 WriteTxtBloc (table[4]); test 3 WriteSegBloc (table[4]); test 4  //TestWriteAlphabet.ino #include &lt;Wire.h&gt;</pre>	<i>Command summary</i>																																																			
	<table border="1"> <thead> <tr> <th>Command</th> <th>Read</th> <th>Write</th> </tr> </thead> <tbody> <tr> <td><b>0 1b</b></td> <td><b>Id=0xC0</b></td> <td></td> </tr> <tr> <td><b>1 5b</b></td> <td></td> <td><b>Bloc 4 segments</b></td> </tr> <tr> <td><b>2 5b</b></td> <td></td> <td><b>Bloc 4 Ascii ABCD</b></td> </tr> <tr> <td><b>3 1b</b></td> <td></td> <td><b>1 Ascii Scroll</b></td> </tr> <tr> <td><b>4 2b</b></td> <td></td> <td><b>16 bits hexa</b></td> </tr> <tr> <td><b>5 2b</b></td> <td></td> <td><b>16 bits decimal</b></td> </tr> <tr> <td><b>6 1b</b></td> <td></td> <td><b>Mode</b></td> </tr> <tr> <td></td> <td></td> <td><b>0 Hexa</b></td> </tr> <tr> <td></td> <td></td> <td><b>1 Ascii</b></td> </tr> <tr> <td></td> <td></td> <td><b>3 Segments</b></td> </tr> <tr> <td></td> <td></td> <td><b>4 Decimal</b></td> </tr> <tr> <td></td> <td></td> <td><b>8 HighDLowB</b></td> </tr> <tr> <td></td> <td></td> <td><b>16 HighBLowD</b></td> </tr> <tr> <td></td> <td></td> <td><b>32 HighDLowD</b></td> </tr> <tr> <td><b>7 2b</b></td> <td></td> <td><b>2 car AB .. (ascii)</b></td> </tr> <tr> <td><b>8 2b</b></td> <td></td> <td><b>2 car .. CD</b></td> </tr> </tbody> </table>	Command	Read	Write	<b>0 1b</b>	<b>Id=0xC0</b>		<b>1 5b</b>		<b>Bloc 4 segments</b>	<b>2 5b</b>		<b>Bloc 4 Ascii ABCD</b>	<b>3 1b</b>		<b>1 Ascii Scroll</b>	<b>4 2b</b>		<b>16 bits hexa</b>	<b>5 2b</b>		<b>16 bits decimal</b>	<b>6 1b</b>		<b>Mode</b>			<b>0 Hexa</b>			<b>1 Ascii</b>			<b>3 Segments</b>			<b>4 Decimal</b>			<b>8 HighDLowB</b>			<b>16 HighBLowD</b>			<b>32 HighDLowD</b>	<b>7 2b</b>		<b>2 car AB .. (ascii)</b>	<b>8 2b</b>		<b>2 car .. CD</b>
Command	Read	Write																																																		
<b>0 1b</b>	<b>Id=0xC0</b>																																																			
<b>1 5b</b>		<b>Bloc 4 segments</b>																																																		
<b>2 5b</b>		<b>Bloc 4 Ascii ABCD</b>																																																		
<b>3 1b</b>		<b>1 Ascii Scroll</b>																																																		
<b>4 2b</b>		<b>16 bits hexa</b>																																																		
<b>5 2b</b>		<b>16 bits decimal</b>																																																		
<b>6 1b</b>		<b>Mode</b>																																																		
		<b>0 Hexa</b>																																																		
		<b>1 Ascii</b>																																																		
		<b>3 Segments</b>																																																		
		<b>4 Decimal</b>																																																		
		<b>8 HighDLowB</b>																																																		
		<b>16 HighBLowD</b>																																																		
		<b>32 HighDLowD</b>																																																		
<b>7 2b</b>		<b>2 car AB .. (ascii)</b>																																																		
<b>8 2b</b>		<b>2 car .. CD</b>																																																		

The X+Go mobile robot uses the same controller with a larger display. The only difference is the push button is not available. See [www.didel.com/diduino/XplusGo.pdf](http://www.didel.com/diduino/XplusGo.pdf)



**DiTell** uses the same hardware with a 1-wire communication, hex and decimal modes only , no alphanumeric.  
**DiTell** will help your realtime debugging on all your C projects, specially with Tiny processors. Usefulness count, not price.  
See [www.didel.com/diduino/DiTelli.pdf](http://www.didel.com/diduino/DiTelli.pdf)



jdn 160314