Grove 4-digit display - Migrating an Arduino Library to Nspire Python

Hans-Martin Hilbig

T³ DEUTSCHLAND

Teachers Teaching with Technology™

Overview

Arduino, based on C++, is a popular programming language in the Maker Space and at Schools. Numerous low-cost sensors are available to be used with Arduino Microcontroller boards. Driver libraries are available from Open-Source portals like github.com. Although syntactically different, Python shares with Arduino its object orientation, its Open-Source nature and its popularity.
Different from Arduino, Python is an interpreter-based language, making it particularly easy for young students to create and debug their code. As an integrated learning system with all applications from coding, debug, data analysis and graphical display in one program, the TI-Innovator System ensures teachers and students focusing on the coding topic instead of infrastructure issues in their STEM (MINT) courses at school.
This project shows the process of migrating an existing Arduino driver library for a non-trivial peripheral like the Grove 4-digit display to a driver library for the TI-Innovator Hub Microcontroller module as part of the TI-Innovator system.

The Grove 4-digit display module

The main components of the Grove 4-digit display module are a multiplexed 4-digits-wide common Anode seven-segment numeric display and a special silicon chip called TM1637, made by Titan Microelectronics [1]. The TM1673 is providing a two-pin MCU digital interface, data latch and LED output drivers. Although the MCU interface is similar to the popular 2-pin I2C standard, its protocol is slightly different and needs a custom software code to control. The major industrial applications of the 4-digit display are induction cookers, micro-wave ovens and other small household electrical appliances.

The 4-digit Arduino driver library

There are multiple 4-digit display driver libraries available on the Internet. In this project, the TM1637Display.h and TM1637.cpp libraries created by Avishorp have been used [2]. Three aspects have been the major reasons for this choice: First, the library is object oriented, allowing multiple instances of the 4-digit display to be used in parallel. Second, the library provides a broad suite of methods to be used, from full custom segment code creation, display of positive and negative decimals, all the way to the display of hexadecimal numbers. Third, the library was written in a strict hierarchical manner, with lower-level methods being used by higher-level methods. This ensures the major migration effort to be focused on the hardware abstraction layer itself, with all other layers ‘only’ needing the translation effort from Arduino to Python syntax.

The 4-digits Arduino hardware interface

The 4-digit display [3] is connected to an Arduino board via a 4-pin Grove cable. Pin 1 is GND, pin 2 Vdd (3.6V-5V), pin 3 is the dio pin and pin 4 the clk pin. Similar to I2C, pin 3 and pin 4 have pullup resistors connected to Vdd. This allows a bi-directional use of the dio pin being a data input as well as a data output for handshaking/acknowledge purposes.To cope with this kind of hardware interface, the signal pin of the Arduino board is either set to an active low output, to provide a logic ‘0’ to the clk/dio pins. Or, the signal pin is programmed to act as a high impedance input, using the pinmode() function in Arduino. In this case, the input would be pulled up to Vdd via the integrated pullup resistors of the Grove board. The Arduino board would be able to read the state of the acknowledge signal on the dio-pin. Different to the TI-Innovator Hub, the Arduino pinmode(),digitalRead() and digitalWrite() functions are almost identical in code execution speed and are able to be toggled in the 3-digit kHz range, at least. This ensures a fast data communication interface between the Microcontroller and the Grove board but requires the use of the Acknowledge-signal on the dio-pin of the TM1637, to ensure proper data transmission between Microcontroller and Grove board.

The 4-digits TI-Innovator Hub hardware interface

Under firmware version 1.4, the TI-Hub is able to toggle its output pins at about 350Hz, when the bbport() function is used. As its name suggests, bbport() can only be used with the BB port pins of the TI-Hub. To allow users to use all features of the 4-digit display driver library, under firmware 1.4 already, an interface using BB1 for the clk-signal and BB2 for the dio-signal has been implemented, limiting the number of displays supported to one.
A digital.in()/digital.out() function call is yielding only about 10-15 samples per second. A direct replication of the Arduino pinmode() I/O switching function in the TI-Hub under 1.4 would result in a very slow update of the 4-digit display, taking more than a second at best.
Along with the release of firmware 1.5, the digital.out() function will be much faster, supporting 'OUT1' and 'OUT 2' as the preferred hardware interface for the 4-digit display. This ensures two Grove displays can be used also in TI-Rover applications, while all BB pins are consumed by the TI-Rover communication interface.
The TM1637's maximum clk input frequency is specified at 500kHz, with data setup and hold times in the < 500ns range. The TI-Hub speed is more than a factor of thousand slower than the maximum speed of the TM1637 to receive data, so a compromise is taken to leave the TI-Hub pins always as outputs to the Grove board, without checking the acknowledge signal for a successful data transmission.

The hardware abstraction layer of the TI-Hub driver library under firmware 1.5

Communication between the Microcontroller and the Grove display is based on an 8-bit protocol, requiring 8 clock cycles per packet. Each clock cycle is composed of two digital.out() function calls, one to set the output to a low level, followed by a second one to set the output to a high level. Setting up the desired command, the address of data being written to and the data-bytes themselves easily can reach 48 clock cycles for a 4-digit display update. One digital.out() function call takes about 2.8ms to execute in Python.
When OUT 1 and OUT 2 are being used, clk- and dio-pins have to be controlled sequentially, through separate digital.out() statements. A 48 cycle clk/dio packet would need 48*2*2*2.8ms = 538ms to execute a full display update cycle, which is pretty slow.
To improve performance in spite of this hardware limitation, the Python code of the hardware abstraction layer has to be optimized for speed. Figure 1 shows the code of the write_port() function. The key element for speed optimization is to execute a digital.out() statement only when there is a change in logic level versus the data sent in the previous write_port() function call. This is accomplished by doing a bit-wise exclusive-OR test of the current clk/dio data pair versus the previous data pair. Any change in logic level would trigger a digital.out() function call for that data bit. In turn, no change in logic level would skip the digital.out() function call for that bit. This results in an average of 2x speed improvement over a conventional coding technique with addressing both signal pins each time data is written to.
Figure 2 shows the display update cycle execution speed for each number from zero to 100. Speed ranges between 250-295ms, depending on the number of segments to update.

TM1637driver1.py

```
def write_port(self,bit_state):
    if self._bb:
        self.mybb.write_port(bit_state)
    else:
        if self._last_bit_state^bit_state == 0b11:
            self.mydio.set((bit_state&0b10)>>1)
            self.myclk.set(bit_state&0b01)
        elif self._last_bit_state^bit_state == 0b10:
            self.mydio.set((bit_state&0b10)>>1)
        elif self._last_bit_state^bit_state == 0b01:
            self.myclk.set(bit_state&0b01)
    self._last_bit_state=bit_state
```

BB port selected?
Just write both bits to BB1/BB2
OUT1 or OUT2 selected?
Did both bits change vs. last call?
Yes, write to the dio port
And to the clk port (2*2.8ms)
Did only the dio bit change?
Write to the dio port (2.8ms)
Did only the clk bit change?
Write to the clk port (2.8ms)
Preserve the current state for the next call.

Figure 1 write_port() function

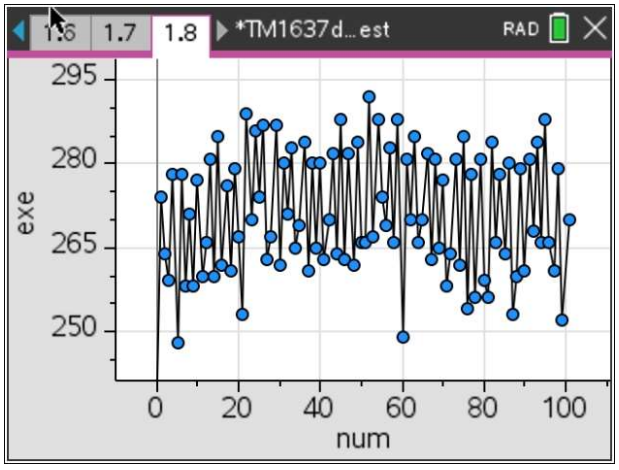

Figure 2 Grove display update speed, counting from 0 to 100

Code comparison – Arduino vs. Python

Figure 3 shows the setSegments() function implementation in Arduino, while figure 4 is showing the exact same function implemented in Python.

TM1637Display

```
void TM1637Display::setSegments(const uint8_t segments[], uint8_t length, uint8_t pos)
{
    // Write COMM1
  start();
  writeByte(TM1637_I2C_COMM1);
  stop();

  // Write COMM2 + first digit address
  start();
  writeByte(TM1637_I2C_COMM2 + (pos & 0x03));

  // Write the data bytes
  for (uint8_t k=0; k < length; k++)
    writeByte(segments[k]);

  stop();

  // Write COMM3 + brightness
  start();
  writeByte(TM1637_I2C_COMM3 + (m_brightness & 0x0f));
  stop();
}
```

Figure 3 setSegments() function in Arduino

TM1637driver.py 81/194

```
def setSegments(self,segments,length=4,pos=0):
    #Write cmd1
    self.start()
    self.writeByte(self._cmd1)
    self.stop()
    #Write cmd2 + first digit address
    self.start()
    self.writeByte(self._cmd2 + (pos & 0x03))
    #Write the data bytes
    for k in range(length):
        self.writeByte(segments[k])
    self.stop()
    #Write cmd3 + brightness
    self.start()
    self.writeByte(self._cmd3 + (self._brightness & 0x0f))
    self.stop()
```

Figure 4 setSegments() function in Python

While Arduino is strict on variable type declaration, this is not needed in Python, as Python is supporting a dynamic variable type casting. Arduino also is strict on formatting symbols such as ';' and '{','}' while Python formatting relies on indentation. But, both programming languages also have a lot of similarities: code density is the same (at least in this example) and arithmetic and logic operators are the same, making it relatively easy to port major portions of code from Arduino to Python and to maintain the same structure and nomenclature of the function names.

Methods and attributes of TM1637 driver library

As described before, all methods are coded in a strictly hierarchical scheme, with a higher level method building on a lower level method. A maximum of one Grove displays is supported under firmware 1.4 and a maximum of three displays supported under 1.5.
An external power supply needs to be connected to the TI-Hub, as the Grove board may draw up to 80mA.
All user level methods can be used with the TM1637display class, as listed in Table 1. The hardware abstraction methods have been listed for completeness and better understanding as well, but a user access of these methods is not recommended.

Method	Abstraction level	Attributes	Defaults	Description
__init__	N/A	Port name (BB, Out 1, Out 2)	N/A	Initializes a TM1637display class object (BB only, under firmware 1.4), max 3 objects under 1.5 and above (Out 1/Out 2/BB)
setBrightness()	User	brightness level (0…6), on/off	level 6 (highest), on	first attribute sets the brightness level of the display, second attribute used to turn the display on or off
setSegments()	User	list of segments, # of digits, start position	# of digits=4, start pos=0	Allows segment level access to the display, for custom character display
clear()	User	N/A	N/A	Clears all segment memory data bits
showNumberDec()	User	number,leading zeroes,# of digits, start position	leading zeroes=off,digits=4, pos=0	Displays pos/neg decimal number at defined length and position, allows filling empty spaces with leading zeroes
showNumberDecEx()	User	base,number,dots, leading zeroes,# of digits, start position	dots=0,leading zeroes=off,digits=4, pos=0	Same as above, but allows setting of a decimal point (if supported by Grove hardware) Some Displays have a colon in the middle of the display and no decimal point
showNumberHexEx()	User	number,dots,leading zeroes,# of digits, start position	dots=0,leading zeroes=off,digits=4, pos=0	Same as above, but displays the value provided in <number> as a hexadecimal number
showNumberBaseEx()	User	base,number,dots, leading zeroes,# of digits, start position	dots=0,leading zeroes=off,digits=4, pos=0	Same as above, but allows setting of the base (10=dec, 16=hex) the value shall be displayed
start()	Hardware	N/A	N/A	Initiates start of a protocol to TM1637
stop()	Hardware	N/A	N/A	Initiates the end of a protocol to TM1637
writeByte()	Hardware	data byte	N/A	sends data byte to TM1637
writeBit()	Hardware	port,value	N/A	writes single bit value to the port defined in <port>
write_port()	Hardware	ADC averaging	avg=3	sends a dio/clk bit-pair to TM1637
show_dots()	Hardware	dots,digits	N/A	function used to display decimal points as part of the showNumberXEx() functions mentioned above
encodeDigit()	Hardware	digit	N/A	digit-to-segment decoder function
ver()	User	N/A	N/A	returns ADXL335 library version info

Table 1 List of callable methods of TM1637driver1.tns library

TM1637driver_demos.tns – demonstrating driver library features

The TM1637driver1_demos.tns Python document contains a couple demo programs to get the user familiar with the methods of the TM1637display() class and to inspire the user to create new experiments using the Grove 4-digit display.

program name	tab #	brief desription
TM1637disptest.py	1.2	Code runs through all the methods of the driver library, providing code examples. The expected display value of each step is displayed in the shell window for comparision
TM1637counter.py	1.4	Code runs in a loop from 0-100, displaying the loop value at max speed, with leading zeroes alternating. A execution speed protocol is created and transferred to a spreadsheet window
two_1637.py	1.6	Code driving 2 displays, one showing pos, one negative loop number. Execution speed is recorded and available in a spreadsheet
minus_test.py	1.8	Code shows a bug in the driver library, not correctly positioning the '-' sign, when leading zeroes are used. This bug is also present in the Arduino library. Feel free to fix it :-)!
tm1637_clock.py	1.10	displays day/time values obtained from localtime(). Use d,s,h controls to change what is displayed
list & spreadsheet window	1.12	shows the speed data recorded by the TM1637counter.py or two_1637.py programs
statistics window	1.13	shows the speed data in graphical form

Table 2 List of demo programs contained in TM1637driver1_demos.tns

Summary:

Besides creating a driver library of the 4-digit Grove display for the TI-Innovator system, this project has been created to demonstrate the steps involved with migrating an existing driver library from Arduino into Python. The biggest migration challenge was the hardware abstraction layer, due to specific requirements and speed limitations of the TI-Hub compared to an Arduino board. Speed optimization of the lowest level hardware driver code has led to acceptable performance, with a minimum of changes (if at all) for the remainder of the library. A big Thank You to Avishorp from Israel for having created a great driver library in Arduino as a base for this project!

Sources:

[1] https://www.mcielectronics.cl/website_MCI/static/documents/Datasheet_TM1637.pdf
[2] https://github.com/avishorp/TM1637
[3] https://www.reichelt.de/de/de/arduino-display-grove-4-zahlen-tm1637-grv-4num-display-p191185.html?r=1