Electronics
More Good Coding Style

Terry Sturtevant

Wilfrid Laurier University

October 3, 2018
Good Coding Style

Good coding style makes programs more readable. It minimizes the use of comments and makes code more easily re-usable.
Good Coding Style

- Good coding style makes programs more readable
Good Coding Style

- Good coding style makes programs more readable
  It minimizes the use of comments
Good Coding Style

- Good coding style makes programs more readable
  - It minimizes the use of comments
  - It makes code more easily re-usable
Good coding style tips:
Good coding style tips:

1. Use consistent case to distinguish variables, constants, etc.
Good coding style tips:

1. Use consistent case to distinguish variables, constants, etc.
2. Use consistent device prefixes in names
Good coding style tips:

1. Use consistent case to distinguish variables, constants, etc.
2. Use consistent device prefixes in names
3. Create self-explanatory variable and function names
Good coding style tips:

1. Use consistent case to distinguish variables, constants, etc.
2. Use consistent device prefixes in names
3. Create self-explanatory variable and function names
4. Don’t use *magic numbers*
PySpidev sample code

```python
import spidev
spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 5000
while True:
    str_val = input("val (0...255, q=quit) : ")
    if str_val == 'q':
        break
    else:
        value = int(str_val)
dummy = spi.xfer2([49, value])
spi.close()
```

Testing the MAX522 DAC

Terry Sturtevant

Electronics More Good Coding Style
PySpidev sample code

```python
import spidev
spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 5000
while True:
    strval = input("val (0...255, q=quit):")
    if strval == 'q':
        break
    else:
        value = int(strval)
        dummy = spi.xfer2([49, value])
spi.close()
```
PySpidev sample code

```python
import spidev
spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 5000
while True:
    strval = input("val (0...255, q=quit):")
    if strval == 'q':
        break
    else:
        value = int(strval)
        dummy = spi.xfer2([49, value])
spi.close()
```

**Testing the MAX522 DAC**
Good Coding Style

Example

```
spi.open(0,0)
→ open port, device
```
spi.open(0,0)

→ open port, device

The Raspberry Pi has only one SPI port available, and it can have two devices.
The Raspberry Pi has only one SPI port available, and it can have two devices.

\[
\begin{align*}
SPI_{PORT} &= 0 \\
SPI_{DEVICE\_DAC} &= 0 \\
\end{align*}
\]

\[
\text{spi.open}(SPI_{PORT},SPI_{DEVICE\_DAC})
\]

This is much clearer.
spi.open(0,0)
→ open port, device

The Raspberry Pi has only one SPI port available, and it can have two devices.

SPI_PORT=0
SPI_DEVICE_DAC=0

spi.open(SPI_PORT, SPI_DEVICE_DAC)

This is much clearer.
Good Coding Style

Example

- \texttt{dummy = spi.xfer2([49,value])}
  \rightarrow \text{transfer bytes}

What is the purpose of ‘49’?
\[ \text{dummy} = \text{spi.xfer2}([49,\text{value}]) \]

→ transfer bytes

What is the purpose of ‘49’? (Note: 49 decimal is 32+16+1)
From the MAX522 datasheet

<table>
<thead>
<tr>
<th>CONTROL</th>
<th>DATA</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>UB1</td>
<td>UB2</td>
<td>UB3</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
</tbody>
</table>

X = Don’t care.
* = Not shown, for the sake of clarity. The functions of loading and shutting down the DACs and programming the logic can be combined in a single command.
From the MAX522 datasheet

### Table 2. Serial-Interface Programming Commands

<table>
<thead>
<tr>
<th>CONTROL</th>
<th>DATA</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>UB1 UB2 UB3 SB SA UB4 LB LA B&lt;sub&gt;7&lt;/sub&gt; MSB B&lt;sub&gt;6&lt;/sub&gt; B&lt;sub&gt;5&lt;/sub&gt; B&lt;sub&gt;4&lt;/sub&gt; B&lt;sub&gt;3&lt;/sub&gt; B&lt;sub&gt;2&lt;/sub&gt; B&lt;sub&gt;1&lt;/sub&gt; B&lt;sub&gt;0&lt;/sub&gt; LSB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X X 1 * * 0 0 0</td>
<td>X X X X X X X</td>
<td>No Operation to DAC Registers</td>
</tr>
<tr>
<td>X X 1 * * 0 0 0</td>
<td></td>
<td>Unassigned Command</td>
</tr>
<tr>
<td>X X 1 * * 0 1 0</td>
<td></td>
<td>8-Bit DAC Data</td>
</tr>
<tr>
<td>X X 1 * * 0 0 1</td>
<td></td>
<td>Load Register to DAC A</td>
</tr>
<tr>
<td>X X 1 * * 0 1 1</td>
<td></td>
<td>Load Register to DAC B</td>
</tr>
<tr>
<td>X X 1 0 0 0 * *</td>
<td>X X X X X X X</td>
<td>Load Both DAC Registers</td>
</tr>
<tr>
<td>X X 1 0 0 0 * *</td>
<td>X X X X X X X</td>
<td>All DACs Active</td>
</tr>
<tr>
<td>X X 1 0 0 0 * *</td>
<td>X X X X X X X</td>
<td>Unassigned Command</td>
</tr>
<tr>
<td>X X 1 1 0 0 0 * *</td>
<td>X X X X X X X</td>
<td>Shut Down DAC B</td>
</tr>
<tr>
<td>X X 1 0 1 0 * *</td>
<td>X X X X X X X</td>
<td>Shut Down DAC A</td>
</tr>
<tr>
<td>X X 1 1 1 0 * *</td>
<td>X X X X X X X</td>
<td>Shut Down All DACs</td>
</tr>
</tbody>
</table>

X = Don't care.

* = Not shown, for the sake of clarity. The functions of loading and shutting down the DACs and programming the logic can be combined in a single command.
From the MAX522 datasheet

Table 2. Serial-Interface Programming Commands

<table>
<thead>
<tr>
<th>CONTROL</th>
<th>DATA</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>UB1</td>
<td>UB2</td>
<td>UB3</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
</tbody>
</table>

X = Don't care.

* = Not shown, for the sake of clarity. The functions of loading and shutting down the DACs and programming the logic can be combined in a single command.

\[32 + 16\]
From the MAX522 datasheet

![Table 2. Serial-Interface Programming Commands](image)

32+16+1

**Terry Sturtevant**  
Electronics More Good Coding Style
Good Coding Style

Example

```
DAC_LOAD_A = 1
DAC_SHUTDOWN_B = 48
```

dummy = spi.xfer2([DAC_LOAD_A + DAC_SHUTDOWN_B, value])

This is much clearer.

Terry Sturtevant

Electronics More Good Coding Style
DAC_LOAD_A=1
DAC_SHUTDOWN_B=48
dummy = spi.xfer2([DAC_LOAD_A + DAC_SHUTDOWN_B, value])

This is much clearer.
A function can help.
Good Coding Style

Example

A function can help.

```python
def outputDAC(spidevice, command, data):
    dummy = spidevice.xfer2(command, data)
    return
```
A function can help.

```python
def outputDAC(spidevice, command, data):
    dummy = spidevice.xfer2(command, data)
    return

outputDAC(spi, DAC_LOAD_A + DAC_SHUTDOWN_B, value)
```
Good Coding Style

Example

PySpidev revised sample code

import spidev

% SPI definitions
SPI PORT=0
SPI DEVICE DAC=0

% DAC definitions
DAC COMPONENT="MAX522"
DAC LOAD A=1
DAC SHUTDOWN B$=48

% definition section
Terry Sturtevant

Electronics More Good Coding Style
import spidev

% SPI definitions
SPI_PORT=0
SPI_DEVICE_DAC=0

% DAC definitions
DAC_COMPONENT="MAX522"
DAC_LOAD_A=1
DAC_SHUTDOWN_B$=48

def outputDAC(spidevice, command, data):
    dummy = spidevice.xfer2(command, data)
    return
PySpidev revised sample code

```python
import spidev

% SPI definitions
SPI_PORT=0
SPI_DEVICE_DAC=0

% DAC definitions
DAC_COMPONENT="MAX522"
DAC_LOAD_A=1
DAC_SHUTDOWN_B$=48

% def outputDAC(spidevice, command, data):
dummy = spidevice.xfer2(command, data)
return

Definition section
```
From the definition section:
From the definition section:

- I can search through all my programs on SPI
From the definition section:

- I can search through all my programs on **SPI**
- By making the **DAC_COMPONENT** definition I can search through all my programs for this specific device, i.e. “MAX522”
From the definition section:

- I can search through all my programs on SPI
- By making the DAC_COMPONENT definition I can search through all my programs for this specific device, i.e. “MAX522”
- I can search this program on DAC to find anything related to this device
From the definition section:

- I can search through all my programs on SPI
- By making the `DAC_COMPONENT` definition I can search through all my programs for this specific device, i.e., “MAX522”
- I can search this program on DAC to find anything related to this device
- The command definitions like `DAC_LOAD_A` make the *purpose* of the code obvious
<table>
<thead>
<tr>
<th>Good Coding Style</th>
<th>Example</th>
</tr>
</thead>
</table>

**PySpidev revised sample code**

```python
spi = spidev.SpiDev()
spi.open(SPI_PORT, SPI_DEVICE_DAC)
spi.max_speed_hz = 5000

while True:
    str_val = input("val (0...255, q=quit): ")
    if str_val == 'q':
        break
    else:
        value = int(str_val)
        outputDAC(spi, DAC_LOAD + DAC_SHUTDOWN, value)

spi.close()
```

**Operation section**

(Note: if getting input from the user were made into a function, this could be simpler still.)

Terry Sturtevant

Electronics More Good Coding Style
PySpidev revised sample code

```python
spi = spidev.SpiDev()
spi.open(SPI_PORT, SPI_DEVICE_DAC)
spi.max_speed_hz = 5000
while True:
    strval = input("val (0...255, q=quit):")
    if strval == 'q':
        break
    else:
        value = int(strval)
        outputDAC(spi, DAC_LOAD_A+DAC_SHUTDOWN_B, value)
spi.close()
```

Operation section
(Note: if getting input from the user were made into a function, this could be simpler still.)
PySpidev revised sample code

```python
spi = spidev.SpiDev()
spi.open(SPI_PORT, SPI_DEVICE_DAC)
spi.max_speed_hz = 5000
while True:
    strval = input("val (0...255, q=quit):")
    if strval == 'q':
        break
    else:
        value = int(strval)
        outputDAC(spi, DAC_LOAD_A+DAC_SHUTDOWN_B, value)
spi.close()
```

Operation section
PySpidev revised sample code

```python
spi = spidev.SpiDev()
spi.open(SPI.PORT, SPI.DEVICE_DAC)
spi.max_speed_hz = 5000
while True:
    strval = input("val (0...255, q=quit):")
    if strval == 'q':
        break
    else:
        value = int(strval)
        outputDAC(spi, DAC_LOAD_A+DAC_SHUTDOWN_B, value)
spi.close()
```

**Operation section** (Note: if getting input from the user were made into a function, this could be simpler still.)