Electronics
More Good Coding Style

Terry Sturtevant

Wilfrid Laurier University

October 3, 2018

Good Coding Style

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

PySpidev sample code

```
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

```
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

spi.open(0,0)

$\rightarrow$ open port, device

spi.open(0,0)

$\rightarrow$ open port, device

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

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)

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.

- **dummy = spi.xfer2([49,value])**

  → transfer bytes

What is the purpose of '49'?

- **dummy = spi.xfer2([49,value])**

  → transfer bytes

What is the purpose of '49'? (Note: 49 decimal is 32+16+1)

From the MAX522 datasheet

Table 2. Serial-Interface Programming Commands

CONTROL								DATA								FUNCTION
UB1	UB2	UB3	SB	SA	UB4	LB	LA	B7 MSB	B6	B5	B4	B3	B2	B1	B0 LSB	
X	X	1	*	*	0	0	0	X	X	X	X	X	X	X	X	No Operation to DAC Registers
X	X	1	*	*	0	0	0									Unassigned Command
X	X	1	*	*	0	1	0	8-Bit DAC Data								Load Register to DAC B
X	X	1	*	*	0	0	1	8-Bit DAC Data								Load Register to DAC A
X	X	1	*	*	0	1	1	8-Bit DAC Data								Load Both DAC Registers
X	X	1	0	0	0	*	*	X	X	X	X	X	X	X	X	All DACs Active
X	X	1	0	0	0	*	*	X	X	X	X	X	X	X	X	Unassigned Command
X	X	1	1	0	0	*	*	X	X	X	X	X	X	X	X	Shut Down DAC B
X	X	1	0	1	0	*	*	X	X	X	X	X	X	X	X	Shut Down DAC A
X	X	1	1	1	0	*	*	X	X	X	X	X	X	X	X	Shut Down All DACs

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

CONTROL								DATA								FUNCTION
UB1	UB2	UB3	SB	SA	UB4	LB	LA	B7 MSB	B6	B5	B4	B3	B2	B1	B0 LSB	
X	X	1	*	*	0	0	0	X	X	X	X	X	X	X	X	No Operation to DAC Registers
X	X	1	*	*	0	0	0									Unassigned Command
X	X	1	*	*	0	1	0	8-Bit DAC Data								Load Register to DAC B
X	X	1	*	*	0	0	1	8-Bit DAC Data								Load Register to DAC A
X	X	1	*	*	0	1	1	8-Bit DAC Data								Load Both DAC Registers
X	X	1	0	0	0	*	*	X	X	X	X	X	X	X	X	All DACs Active
X	X	1	0	0	0	*	*	X	X	X	X	X	X	X	X	Unassigned Command
X	X	1	1	0	0	*	*	X	X	X	X	X	X	X	X	Shut Down DAC B
X	X	1	0	1	0	*	*	X	X	X	X	X	X	X	X	Shut Down DAC A
X	X	1	1	1	0	*	*	X	X	X	X	X	X	X	X	Shut Down All DACs

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

CONTROL								DATA								FUNCTION
UB1	UB2	UB3	SB	SA	UB4	LB	LA	B7 MSB	B6	B5	B4	B3	B2	B1	B0 LSB	
X	X	1	*	*	0	0	0	X	X	X	X	X	X	X	X	No Operation to DAC Registers
X	X	1	*	*	0	0	0									Unassigned Command
X	X	1	*	*	0	1	0	8-Bit DAC Data								Load Register to DAC B
X	X	1	*	*	0	0	1	8-Bit DAC Data								Load Register to DAC A
X	X	1	*	*	0	1	1	8-Bit DAC Data								Load Both DAC Registers
X	X	1	0	0	0	*	*	X	X	X	X	X	X	X	X	All DACs Active
X	X	1	0	0	0	*	*	X	X	X	X	X	X	X	X	Unassigned Command
X	X	1	1	0	0	*	*	X	X	X	X	X	X	X	X	Shut Down DAC B
X	X	1	0	1	0	*	*	X	X	X	X	X	X	X	X	Shut Down DAC A
X	X	1	1	1	0	*	*	X	X	X	X	X	X	X	X	Shut Down All DACs

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

CONTROL								DATA								FUNCTION
UB1	UB2	UB3	SB	SA	UB4	LB	LA	B7 MSB	B6	B5	B4	B3	B2	B1	B0 LSB	
X	X	1	*	*	0	0	0	X	X	X	X	X	X	X	X	No Operation to DAC Registers
X	X	1	*	*	0	0	0									Unassigned Command
X	X	1	*	*	0	1	0	8-Bit DAC Data								Load Register to DAC B
X	X	1	*	*	0	0	1	8-Bit DAC Data								Load Register to DAC A
X	X	1	*	*	0	1	1	8-Bit DAC Data								Load Both DAC Registers
X	X	1	0	0	0	*	*	X	X	X	X	X	X	X	X	All DACs Active
X	X	1	0	0	0	*	*	X	X	X	X	X	X	X	X	Unassigned Command
X	X	1	1	0	0	*	*	X	X	X	X	X	X	X	X	Shut Down DAC B
X	X	1	0	1	0	*	*	X	X	X	X	X	X	X	X	Shut Down DAC A
X	X	1	1	1	0	*	*	X	X	X	X	X	X	X	X	Shut Down All DACs

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.*

$32+16+1$

DAC_LOAD_A=1

DAC_SHUTDOWN_B=48

```
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.

A function can help.

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

A function can help.

```
def outputDAC(spidevice, command, data):
    dummy = spidevice.xfer2(command,data)
    return
outputDAC(spi,DAC_LOAD_A +
DAC_SHUTDOWN_B,value)
```

PySpidev revised sample code

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

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
%
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

PySpidev revised sample code

PySpidev revised sample code

```
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()
```

PySpidev revised sample code

```
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

```
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.)