Technical Report: Automated Bandgap Reference Voltage Trimming

1. Objective

This report documents an automated Python-based procedure for trimming the bandgap reference voltage of a mixed-signal IC using I²C communication. The script sweeps a 4-bit trim code, measures the resulting voltage, and selects the optimal code to minimize error against a target value. The implementation is hardware-agnostic and supports integration with real measurement equipment.

---

2. Environment Setup

2.1. Installing the `dfttools` Library

To ensure compatibility, uninstall any existing version of `dfttools` and reinstall the latest from the official GitHub repository:

```
pip uninstall -y dfttools
pip install git+https://github.com/HarishkumarSedu/dfttools.git@main
```

---

3. Complete Bandgap Trimming Code

Below is the full Python code for the bandgap trimming procedure, including hardware callback integration.

```python
from dfttools import *
from time import sleep
from PyMCP2221A import PyMCP2221A
from multimeter import mul_34401A

Test_Name = 'Trim_BG'
print(f'............ {Test_Name} ........')

# I2C device and register setup
I2C_WRITE(device_address=0x68, field_info={'fieldname': 'i2c_page_sel', 'length': 1, 'registers': [{'REG': '0xFE', 'POS': 0, 'RegisterName': 'Page selection', 'RegisterLength': 8, 'Name': 'i2c_page_sel', 'Mask': '0x1', 'Length': 1, 'FieldMSB': 0, 'FieldLSB': 0, 'Attribute': '0000000N', 'Default': '00', 'User': '000000YY', 'Clocking': 'SMB', 'Reset': 'C', 'PageName': 'PAG0'}]}, write_value=0x0)
I2C_WRITE(device_address=0x68, field_info={"fieldname": "i2c_unlock", "length": 1, "registers": [{"REG": "0x00", "POS": 0, "RegisterName": "Config REG1", "RegisterLength": 8, "Name": "i2c_unlock", "Mask": "0x1", "Length": 1, "FieldMSB": 0, "FieldLSB": 0, "Attribute": "NNNNNNNN", "Default": "00", "User": "000YYYYY", "Clocking": "FRO", "Reset": "C", "PageName": "PAG0"}]}, write_value=0x1)
```

```
I2C_WRITE(device_address=0x68, field_info={"fieldname": "tdm_fsyn_rate_mnt_en",
"length": 1, "registers": [{"REG": "0x55", "POS": 7, "RegisterName": "Clock
monitor settings 1", "RegisterLength": 8, "Name": "tdm_fsyn_rate_mnt_en", "Mask":
"0x80", "Length": 1, "FieldMSB": 7, "FieldLSB": 7, "Attribute": "NNNNNNNN",
"Default": "E1", "User": "YYYYYYYY", "Clocking": "FRO", "Reset": "C", "PageName":
"PAG0"}]}, write_value=0x1)
I2C_WRITE(device_address=0x68, field_info={'fieldname': 'unlock_tst_addr',
'length': 1, 'registers': [{'REG': '0xF7', 'POS': 0, 'RegisterName': 'Unlock
register', 'RegisterLength': 8, 'Name': 'unlock_tst_addr', 'Mask': '0x1',
'Length': 1, 'FieldMSB': 0, 'FieldLSB': 0, 'Attribute': '0000000R', 'Default':
'00', 'User': '00000000', 'Clocking': 'FRO', 'Reset': 'C', 'PageName': 'PAG0'}]},
write_value=0xaa)
I2C_WRITE(device_address=0x68, field_info={'fieldname': 'unlock_tst_addr',
'length': 1, 'registers': [{'REG': '0xF7', 'POS': 0, 'RegisterName': 'Unlock
register', 'RegisterLength': 8, 'Name': 'unlock_tst_addr', 'Mask': '0x1',
'Length': 1, 'FieldMSB': 0, 'FieldLSB': 0, 'Attribute': '0000000R', 'Default':
'00', 'User': '00000000', 'Clocking': 'FRO', 'Reset': 'C', 'PageName': 'PAG0'}]},
write_value=0xbb)
I2C_WRITE(device_address=0x68, field_info={'fieldname': 'i2c_page_sel', 'length':
2, 'registers': [{'REG': '0xFE', 'POS': 0, 'RegisterName': 'Page selection',
'RegisterLength': 8, 'Name': 'i2c_page_sel', 'Mask': '0x1', 'Length': 1,
'FieldMSB': 0, 'FieldLSB': 0, 'Attribute': '0000000N', 'Default': '00', 'User':
'000000YY', 'Clocking': 'SMB', 'Reset': 'C', 'PageName': 'PAG0'}]},
write_value=0x1)
I2C_WRITE(device_address=0x68, field_info={'fieldname': 'otp_burnt_b', 'length':
1, 'registers': [{'REG': '0x40', 'POS': 7, 'RegisterName': 'OTP register 0',
'RegisterLength': 8, 'Name': 'otp_burnt_b', 'Mask': '0x80', 'Length': 1,
'FieldMSB': 7, 'FieldLSB': 7, 'Attribute': 'NNNNNNNN', 'Default': '80', 'User':
'00000000', 'Clocking': 'FRO', 'Reset': 'C', 'PageName': 'PAG1'}]},
write_value=0x0)
I2C_WRITE(device_address=0x68, field_info={'fieldname': 'amux2_en', 'length': 1,
'registers': [{'REG': '0x20', 'POS': 1, 'RegisterName': 'Analog test 3',
'RegisterLength': 8, 'Name': 'amux2_en', 'Mask': '0x2', 'Length': 1, 'FieldMSB':
1, 'FieldLSB': 1, 'Attribute': 'NNNNNNNN', 'Default': '00', 'User': '0000YYYY',
'Clocking': 'SMB', 'Reset': 'C', 'PageName': 'PAG1'}]}, write_value=0x1)
I2C_WRITE(device_address=0x68, field_info={'fieldname': 'ref_test_en', 'length':
1, 'registers': [{'REG': '0x1E', 'POS': 5, 'RegisterName': 'Analog test 1',
'RegisterLength': 8, 'Name': 'ref_test_en', 'Mask': '0x20', 'Length': 1,
'FieldMSB': 5, 'FieldLSB': 5, 'Attribute': 'NNNNNNNN', 'Default': '00', 'User':
'00000000', 'Clocking': 'SMB', 'Reset': 'C', 'PageName': 'PAG1'}]},
write_value=0x1)
I2C_WRITE(device_address=0x68, field_info={"fieldname": "ana_test_sel", "length":
4, "registers": [{"REG": "0x1F", "POS": 0, "RegisterName": "Analog test 2",
"RegisterLength": 8, "Name": "ana_test_sel[3:0]", "Mask": "0xF", "Length": 4,
"FieldMSB": 3, "FieldLSB": 0, "Attribute": "NNNNNNNN", "Default": "00", "User":
"00000000", "Clocking": "SMB", "Reset": "C", "PageName": "PAG1"}]},
write_value=0x6)

# Trimming parameters
percentage = 0.1 # 10% difference
typical_value = 1.242
low_value = typical_value - typical_value*percentage
high_value = typical_value + typical_value*percentage
step_size = 0.077625 # mV
```

```
num_steps = 2**4  # 4-bit
noise_std_dev = 0.025

min_error = float('inf')
optimal_code = None
optimal_measured_value = None

# Sweep trim codes and measure
for i in range(num_steps):
    I2C_WRITE(device_address=0x68, field_info={'fieldname': 'ref_vbg_trim',
'length': 4, 'registers': [{'REG': '0xC1', 'POS': 4, 'RegisterName': 'OTP register
129', 'RegisterLength': 8, 'Name': 'ref_vbg_trim[3:0]', 'Mask': '0xF0', 'Length':
4, 'FieldMSB': 3, 'FieldLSB': 0, 'Attribute': 'NNNNNNNN', 'Default': '00', 'User':
'00000000', 'Clocking': 'FRO', 'Reset': 'C', 'PageName': 'PAG1'}]},
write_value=hex(i))
    expected_value = low_value + i * step_size
    measured_value = VMEASURE(signal="HWMute", reference="GND",
expected_value=expected_value, error_spread=noise_std_dev)
    error = abs(measured_value - typical_value)/abs(typical_value) * 100
    if error < min_error:
        min_error = error
        optimal_code = hex(i)
        optimal_measured_value = measured_value
    sleep(0.1)

# Validate and program result
if low_value < optimal_measured_value < high_value:
    print(f'............ {Test_Name} Passed ........')
    I2C_WRITE(device_address=0x68, field_info={'fieldname': 'ref_vbg_trim',
'length': 4, 'registers': [{'REG': '0xC1', 'POS': 4, 'RegisterName': 'OTP register
129', 'RegisterLength': 8, 'Name': 'ref_vbg_trim[3:0]', 'Mask': '0xF0', 'Length':
4, 'FieldMSB': 3, 'FieldLSB': 0, 'Attribute': 'NNNNNNNN', 'Default': '00', 'User':
'00000000', 'Clocking': 'FRO', 'Reset': 'C', 'PageName': 'PAG1'}]},
write_value=optimal_code)
else:
    print(f'............ {Test_Name} Failed ........')
    I2C_WRITE(device_address=0x68, field_info={'fieldname': 'ref_vbg_trim',
'length': 4, 'registers': [{'REG': '0xC1', 'POS': 4, 'RegisterName': 'OTP register
129', 'RegisterLength': 8, 'Name': 'ref_vbg_trim[3:0]', 'Mask': '0xF0', 'Length':
4, 'FieldMSB': 3, 'FieldLSB': 0, 'Attribute': 'NNNNNNNN', 'Default': '00', 'User':
'00000000', 'Clocking': 'FRO', 'Reset': 'C', 'PageName': 'PAG1'}]}, write_value=0)

print(f"Optimal Code: {optimal_code}")
print(f"Optimal measured value : {optimal_measured_value}V, Target value :
{typical_value}V")
print(f"Minimum Error: {min_error}%")

# Hardware callback functions
def custom_i2c_write_callback(device_address: int, register_address: int, value:
int, register):
    mcp2221 = PyMCP2221A.PyMCP2221A()
    mcp2221.I2C_Init()
    default = int(register['Default'], 16)
    mask = int(register['Mask'], 16)
```

```
    LSB = register['POS']
    print(f"Writing {hex(value)} to device {hex(device_address)}, register
{hex(register_address)},")
    value = ((default & mask) | value << LSB) & 0xFF
    mcp2221.I2C_Write(device_address, bytearray([register_address, value]))
    return True

def custom_voltage_measure_callback(signal, reference):
    multimeter = mul_34401A('USB0::0x2A8D::0x1401::MY57229870::INSTR')
    if (Voltage := multimeter.meas_V()):
        measure_hardware_available = True
        return measure_hardware_available, Voltage
    else:
        measure_hardware_available = False
        return measure_hardware_available, None

g.hardware_callbacks = {
    'i2c_write': custom_i2c_write_callback,
    'voltage_measure': custom_voltage_measure_callback,
}

def trail():
    import Trim_BG as Trim_BG

if __name__ == "__main__":
    trail()
```

---

4. Technical Notes

- **Register Operations:** All I^2C register writes use detailed field information, ensuring only the intended bits are modified.
- **Trim Sweep:** The script sweeps all 16 possible trim codes, measuring the resulting bandgap voltage each time.
- **Measurement:** The `VMEASURE` function can be simulated or connected to a real multimeter via VISA.
- **Callbacks:** Hardware operations are abstracted via callbacks for flexibility and testability.
- **Result Validation:** The script checks if the trimmed value is within ±10% of the target and writes the optimal code or a safe default accordingly.

---

5. Sample Output

```
............ Trim_BG ........
Writing 0x0 to device 0x68, register 0xfe,
Writing 0x1 to device 0x68, register 0x0,
...
Writing 0xd to device 0x68, register 0xc1,
Optimal Code: 0xd
Optimal measured value : 1.2318761V, Target value : 1.242V
Minimum Error: 0.8151288244766476%
```

---

6. Conclusion

This script provides a robust, automated solution for bandgap reference voltage trimming, supporting both simulation and real hardware. The code is modular, maintainable, and ready for integration into production test flows.
