DesignSpark.Pmod Documentation

Release 0.2.0

RS Components Ltd

Feb 26, 2020

Contents

1 Supported Pmods 3

2 Installation 5

3 Documentation 7

4 Table of Contents 9

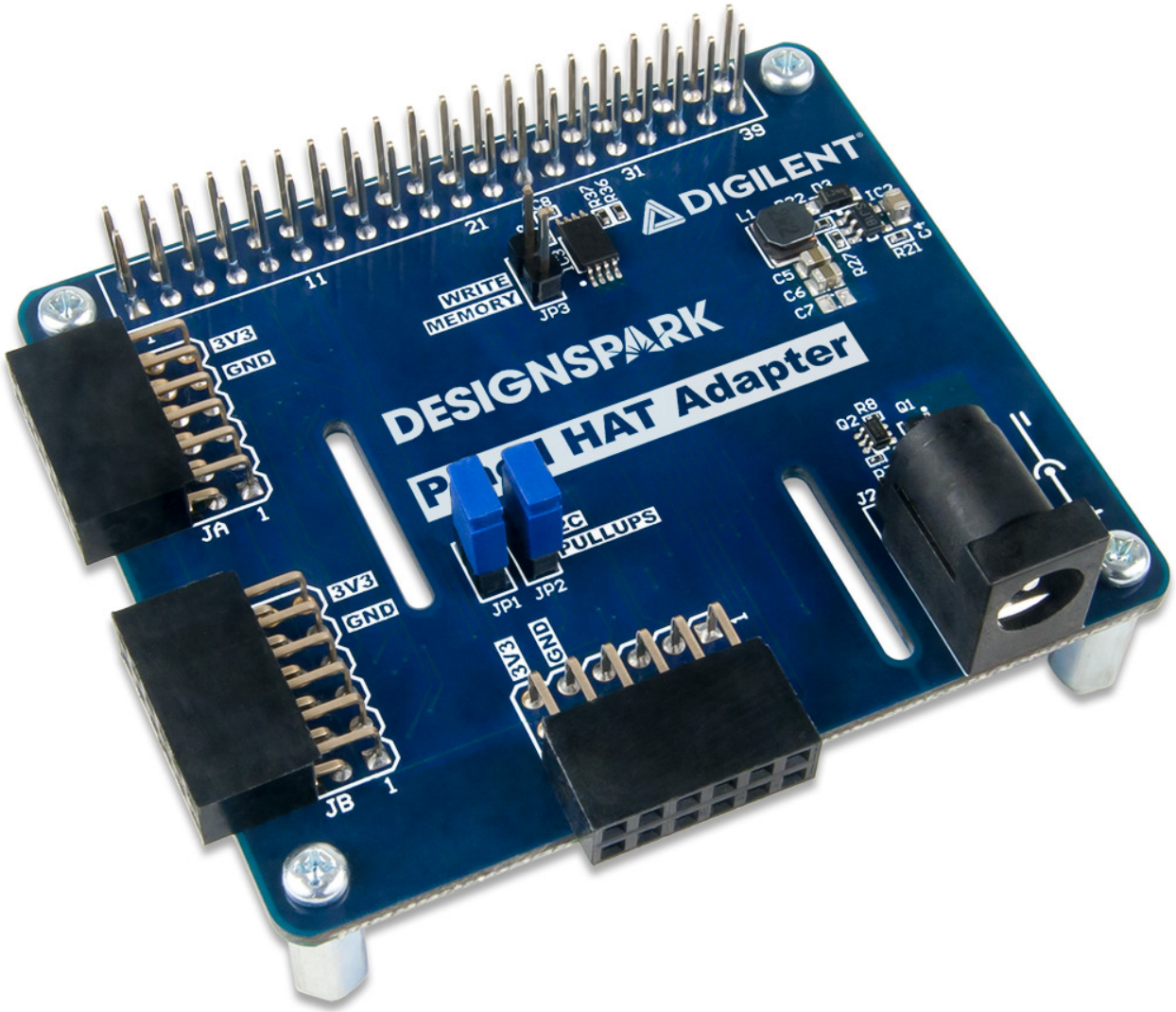
5 Change log 37

6 The MIT License (MIT) 39

Python Module Index 41

Index 43

DESIGNSPARK Pmod Library



Python library to support using Pmods with a Raspberry Pi and the DesignSpark Pmod HAT.

Features include:

- Simple interfaces for supported Pmods
- Checking that Pmod and port capabilities match

- Checking for port usage conflicts
- Usage examples

CHAPTER 1

Supported Pmods

The following modules are currently supported with the DesignSpark Raspberry Pi Pmod HAT:

- PmodACL2 3-axis Accelerometer
- PmodAD1 12-bit ADC
- PmodGPS GPS Module
- PmodHB3 2A H-bridge Driver
- PmodISNS20 20A Current Sensor
- PmodKYPD 16-Button Keypad
- PmodLS1 Line Follower Sensor Interface
- PmodMIC3 MEMS Microphone Module
- PmodOLEDrgb 96x64 RGB OLED Display [1]
- PmodSWT Four Slides Switches
- PmodTC1 K Type Thermocouple Module with Wire
- PmodACL2 3-axis Accelerometer
- PmodGPS GPS Satellite Receiver
- PmodKYPD 16-Button Keypad
- PmodLS1 Line Follower Sensor
- PmodSWT Four Slides Switches

[1] *Builds on the excellent luma.oled and luma.core libraries from Richard Hull and contributors.*

CHAPTER 2

Installation

DesignSpark.Pmod can be installed from PyPi using pip. See the documentation for details.

CHAPTER 3

Documentation

Installation and API documentation, along with examples, can be found at:

http://designspark-pmod.readthedocs.io

For Pmod HAT documentation, including the reference manual and schematic, see:

https://reference.digilentinc.com/reference/add-ons/pmod-hat/start

CHAPTER 4

Table of Contents

4.1 Installation

This guide assumes that you are running Raspbian Stretch.

First enable SPI and configure the UART:

```
pi@raspberrypi:~$ sudo raspi-config
```

Selecting:

- Option 5 - Interfacing
- P4 - SPI
- Enable → YES
- P6 - Serial
- Would you like a login shell to be accessible over serial? → No
- Would you like the serial port hardware to be enabled? → Yes

Then exit raspi-config.

Next update the package lists:

```
pi@raspberrypi:~$ sudo apt-get update
```

Then install the Raspbian dependencies:

```
pi@raspberrypi:~$ sudo apt-get install python-pip python-dev libfreetype6-dev libjpeg-
↪dev build-essential
```

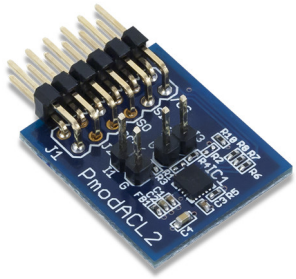
Finally, install DesignSpark.Pmod and dependencies from PyPi:

```
pi@raspberrypi:~$ sudo pip install designspark.pmod
```

4.2 Pmod Information

Details of currently supported Pmods can be found below.

4.2.1 ACL2

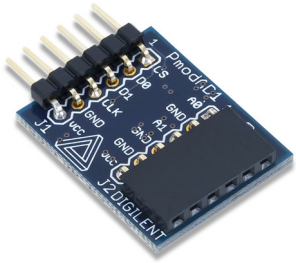


The Pmod ACL2 is a 3-axis MEMS accelerometer powered by the Analog Devices ADXL362. By communicating with the chip via the SPI protocol, users may receive up to 12 bits of resolution for each axis of acceleration. Additionally, this module offers freefall detection as well as power saving features through its motion activated sleep and wake modes.

Features:

- 3-axis MEMS accelerometer
- Up to 12 bits of resolution per axis
- User-selectable resolution
- Activity/inactivity monitoring
- Low current consumption at <2 μA at 100Hz
- Free-fall detection

4.2.2 AD1



The Digilent Pmod AD1 (Revision G) is a two channel 12-bit analog-to-digital converter that features Analog Devices' AD7476A. With a sampling rate of up to 1 million samples per second, this Pmod™ is capable of excelling in even the most demanding audio applications.

Features:

- Two channel 12-bit analog-to-digital converter

- Simultaneous A/D conversion at up to one MSa per channel
- Two 2-pole Sallen-Key anti-alias filters
- Small PCB size for flexible designs 0.95 in × 0.8 in (2.4 cm × 2.0 cm)

Note: Only a single channel (A1) is supported at present due to the way that SPI is configured.

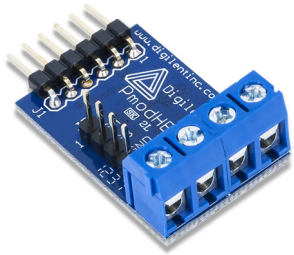
4.2.3 GPS

The Pmod GPS can provide satellite-positioning accuracy to any embedded system. By communicating through UART with the GlobalTop FGPMMOPA6H GPS module, users may benefit from the 3-meter accuracy for any long term traveling.

Features:

- Ultra-sensitive GPS module (-165 dBm)
- Add 3m 2D satellite positioning accuracy to any embedded system
- Low power consumption
- Up to 10Hz update rate
- NMEA (default) and RTCM protocols available

4.2.4 HB3



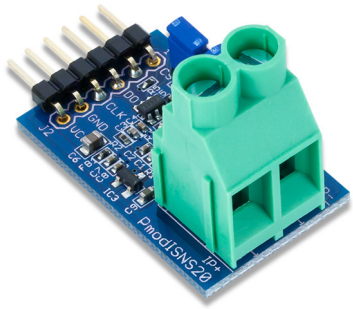
The Pmod HB3 utilizes a full H-Bridge circuit to allow users to drive DC motors from the system board. Two external pins are provided on the Pmod for sensor feedback on the DC motor, if desired.

Features:

- 2A H-bridge circuit

- Drive a DC motor with operation voltage up to 12V
- Screw terminal blocks for connection to the motor
- Separate header for external motor feedback

4.2.5 ISNS20



The Digilent Pmod ISNS20 (Revision A) is a small current sense module with a digital SPI interface. The board combines an Allegro ACS722 Hall Effect current sensor with a 12-bit analog-to-digital converter from Texas Instruments. The Pmod ISNS20 is quick, accurate, and easy to use for a variety of applications.

Features:

- High accuracy current sensor
- Measure current with 120Hz/20kHz/80kHz jumper selections
- $\pm$20A DC or AC input
- Accurate to within $\pm$2%
- 12-bit ADC

4.2.6 KYPD

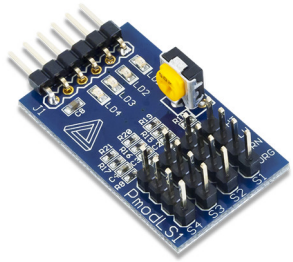
The Pmod KYPD is a 16-button keypad arranged in a hexidecimal format (0-F). By digitally driving a column line to a logic low level and digitally reading each of the rows, users can determine which button is currently pressed.

Features:

- 16 momentary push-buttons
- Can detect simultaneous button presses

- Isolated rows and columns

4.2.7 LS1

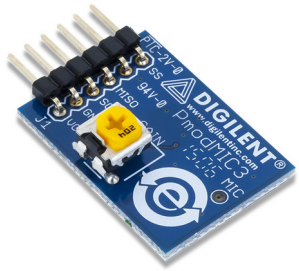


The Digilent Pmod LS1 allows users to receive signals from multiple optical sensors, such as the popular combination of an IR LED with an IR sensor used in line-following robots.

Features:

- Infrared light detector with on-board sensitivity adjustment
- Interface with up to four reflective or transmissive photo detectors
- Works with Digilent IR Proximity Sensor

4.2.8 MIC3

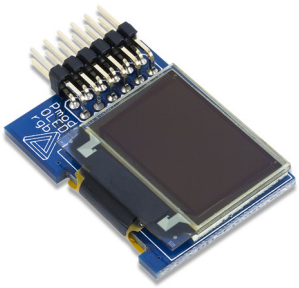


The Digilent Pmod MIC3 (Revision A) is small microphone module with a digital interface. With a Knowles Acoustics SPA2410LR5H-B MEMs microphone and Texas Instrument's ADCS7476 12-bit Analog-to-Digital Converter, you can capture your audio inputs with ease.

Features:

- MEMS Microphone module with digital interface
- Transform audio inputs with 12-bit A/D converter
- Adjust incoming volume with on-board potentiometer
- Up to 1 MSPS of data

4.2.9 OLEDrgb

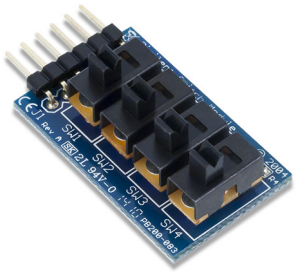


The Digilent Pmod OLEDrgb (Revision B) is an organic RGB LED module with a 96×64 pixel display capable of 16-bit color resolution.

Features:

- 96×64 pixel RGB OLED screen
- 0.8“ x 0.5” graphical display
- 16-bit color resolution
- Two low-power display shutdown modes

4.2.10 SWT

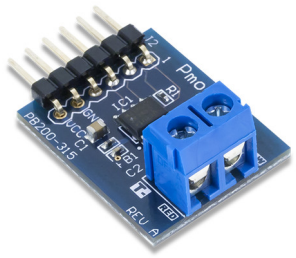


The Pmod SWT provides users with four slide switches for up to 16 different binary logic inputs to for the attached system board.

Features:

- 4 slide switches
- Add user input to host board or project
- Static binary logic input

4.2.11 TC1



The Digilent Pmod TC1 (Revision A) is a cold-junction thermocouple-to-digital converter module designed for a classic K-Type thermocouple wire. With Maxim Integrated's MAX31855, this module reports the measured temperature in 14-bits with 0.25°C resolution.

Features:

- K-type thermocouple-to-digital converter
- Wide temperature range of -73°C to 482°C with provided wire
- ±2°C accuracy from -200°C to 700°C
- 14-bit with 0.25°C resolution
- Cold-junction temperature compensation

4.3 Basic Examples

4.3.1 AD1

12-bit analog-to-digital converter.

Print volts out to the terminal

Read ADC channel A1, print the voltage measured out to the terminal, sleep for 0.8s and repeat.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2017 RS Components Ltd
# SPDX-License-Identifier: MIT License

"""
Read ADC channel A1 and print volts out.
"""

from DesignSpark.Pmod.HAT import createPmod
import time

if __name__ == '__main__':
    adc = createPmod('AD1','JBA')
    time.sleep(0.1)
```

(continues on next page)

(continued from previous page)

```
    try:
        while True:
            volts = adc.readA1Volts()
            print(volts)
            #val = adc.readA1()
            #print(val)
            time.sleep(0.8)
    except KeyboardInterrupt:
        pass
    finally:
        adc.cleanup()
```

Requirements

- PmodAD1 module connected to port JBA
- A voltage source connected to ADC channel A1

4.3.2 HB3

2A H-bridge circuit for DC motor drive up to 12V.

Spin motor

This example:

1. Spins the motor forwards for 20 seconds
2. Commands the motor to stop and pauses for 2 seconds
3. Spins the motor in reverse for 20 seconds
4. Commands the motor to stop and pauses for 2 seconds
5. Ramps up the speed across 100 steps in the forward direction
6. Ramps down the speed across 100 steps in the forward direction
7. Ramps up speed across 100 steps in the reverse direction
8. Ramps down the speed across 100 steps in the reverse direction
9. Loops back to (1)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2017 RS Components Ltd
# SPDX-License-Identifier: MIT License

"""
Spin motor forwards then backwards.
Ramp speed up and then down in forward direction.
Ramp speed up and then down in reverse direction.
"""

from DesignSpark.Pmod.HAT import createPmod
```

(continues on next page)

(continued from previous page)

```
import time

if __name__ == '__main__':

    motor = createPmod('HB3','JAA')

    try:
        while True:

            print('fwd')
            motor.forward(20)
            time.sleep(2)
            motor.stop()
            time.sleep(2)
            print('rev')
            motor.reverse(20)
            time.sleep(1)
            motor.stop()
            time.sleep(2)


            print ('ramp up fwd')
            for i in range(100):
                motor.forward(i)
                time.sleep(.1)

            print ('ramp down fwd')
            for i in range(100):
                motor.forward(100-i)
                time.sleep(.1)

            motor.stop()
            time.sleep(2)

            print ('ramp up rev')
            for i in range(100):
                motor.reverse(i)
                time.sleep(.1)

            print ('ramp down rev')
            for i in range(100):
                motor.reverse(100-i)
                time.sleep(.1)

    except KeyboardInterrupt:
        pass

    finally:
        motor.cleanup()
```

Requirements

- PmodHB3 module connected to port JAA

- DC motor
- Motor power supply

4.3.3 ISNS20

±20A DC or AC input, high accuracy current sensor.

Print milliamps out to the terminal

Read the current sense module, print the milliamps out to the terminal, sleep for 0.8s and repeat.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2017 RS Components Ltd
# SPDX-License-Identifier: MIT License

"""
Read current and print out milliamps.
"""

from DesignSpark.Pmod.HAT import createPmod
import time

if __name__ == '__main__':

    isens = createPmod('ISNS20','JBA')
    time.sleep(0.1)

    try:
        while True:
            mA = isens.readMilliAmps()
            print(mA)
            time.sleep(0.8)
    except KeyboardInterrupt:
        pass
    finally:
        isens.cleanup()
```

Requirements

- PmodISNS20 module connected to port JBA
- Suitable current source, e.g. a power supply and load

4.3.4 MIC3

Knowles Acoustics SPA2410LR5H-B MEMs microphone and Texas Instrument's ADCS7476 12-bit Analog-to-Digital Converter.

Display mic level

Print a continous sound level reading out to the terminal.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2017 RS Components Ltd
# SPDX-License-Identifier: MIT License

"""
Print a continous sound level reading out.
"""

from DesignSpark.Pmod.HAT import createPmod
import time

s = ':'
lut = [s]
for i in range(128):
    s+=':'
    lut.append(s)

if __name__ == '__main__':

    mic = createPmod('MIC3','JBA')
    time.sleep(0.1)

    try:
        while True:
            int = mic.readIntegerValue()
            #print(int)
            print(lut[int>>5])
            snd = mic.readPhysicalValue()
            #print(snd)

            #time.sleep(0.01)
    except KeyboardInterrupt:
        pass
    finally:
        mic.cleanup()
```

Requirements

- PmodMIC3 module connected to port JBA

4.3.5 OLEDrgb

Organic RGB LED module with a 96×64 pixel display capable of 16-bit color resolution.

Display text in a bounding box

Display the text “Hello, World!” in a bounding box.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2017 RS Components Ltd
# SPDX-License-Identifier: MIT License

"""
Display Hello, World! in bounding box.
"""

from DesignSpark.Pmod.HAT import createPmod
from luma.core.render import canvas
from luma.oled.device import ssd1331

if __name__ == '__main__':
    try:
        oled = createPmod('OLEDrgb','JA')
        device = oled.getDevice()

        with canvas(device) as draw:
            draw.rectangle(device.bounding_box, outline="white", fill="black")
            draw.text((16,20), "Hello, World!", fill="white")

        while True:
            pass
    except KeyboardInterrupt:
        pass
    finally:
        oled.cleanup()
```

Luma.Core & Luma.OLED API: `luma.core.render.canvas`, `luma.oled.device.ssd1331`.

Requirements

- PmodOLEDrgb connected to port JA

4.3.6 PmodTC1

A cold-junction thermocouple-to-digital converter module designed for a classic K-Type thermocouple wire. Features a temperature range of -73°C to 482°C with provided wire.

Print celsius out to the terminal

Print the celsius reading out to the terminal, sleep for 0.8s and repeat.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2017 RS Components Ltd
# SPDX-License-Identifier: MIT License

"""
Print the celsius reading out.
"""
```

(continues on next page)

(continued from previous page)

```
from DesignSpark.Pmod.HAT import createPmod
import time

if __name__ == '__main__':

    therm = createPmod('TC1','JBA')
    time.sleep(0.1)

    try:
        while True:
            cel = therm.readCelcius()
            print(cel)
            #intn = therm.readInternal()
            #print(intn)
            time.sleep(0.8)
    except KeyboardInterrupt:
        pass
    finally:
        therm.cleanup()
```

Requirements

- PmodTC1 module connected to port JBA

4.3.7 PmodACL2

A 3-axis MEMS accelerometer module. Features the Analog Devices ADXL362 device with Measurement ranges ±2g, ±4g, ±8g. Up to 12-bit resolution on each axis.

Print axis out to the terminal

Print the axis reading out to the terminal, sleep for 0.5s and repeat.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2020 RS Components Ltd
# SPDX-License-Identifier: MIT License

"""
Read and print out accelerometer values.
"""

from DesignSpark.Pmod.HAT import createPmod
import time

if __name__ == '__main__':

    ACL2 = createPmod('ACL2','JB')
    time.sleep(0.1)

    # print(hex(ACL2.getDeviceID())) # Device test. 0xAD will be expected.
```

(continues on next page)

(continued from previous page)

```
    # ACL2.setRange(ACL2.SENSOR_RANGE_8G) # default setting: +/- 4G
    # maxz = 0

    try:
        while True:
            x,y,z,t = ACL2.getXYZT()
            print(x,y,z,t)
            time.sleep(0.8)
            # if maxz < z:
            #     maxz = z
    except KeyboardInterrupt:
        pass
    finally:
        ACL2.cleanup()
        # print(maxz)
```

Requirements

- PmodACL2 module connected to port JB

4.3.8 PmodGPS

A GlobalTop FGPMMOPA6H GPS antenna module to receive position data from GPS satellites.

Print date, time, and location out to the terminal

Print the date, time, and location reading out to the terminal, sleep for 0.5s and repeat.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2020 RS Components Ltd
# SPDX-License-Identifier: MIT License

"""
Read GPS and print position, speed, and GPS time
"""

from DesignSpark.Pmod.HAT import createPmod
import time
# import webbrowser

if __name__ == '__main__':

    GPS = createPmod('GPS','JCA') # UART port is only available on JCA
    time.sleep(0.1)

    try:
        while True:
            """
            LINE = GPS.getGPSLine()
            print(LINE)
            time.sleep(0.5)
```

(continues on next page)

(continued from previous page)

```
            """
            GPS.gpsUpdate()
            print(GPS.getGPSPosData())
            time.sleep(0.5)
    except KeyboardInterrupt:
        pass
    finally:
        #day,month,year,hour,minute,sec,LatDeg,LogDeg,PDOP = GPS.getGPSPosData()
        #url = "https://google.com/maps?q={},{}"
        #url = url.format(LatDeg,LogDeg)
        GPS.cleanup()
        #webbrowser.open(url)
```

Requirements

- PmodGPS module connected to port JCA
- View of the sky or external antenna connected to the module
- . . . takes time to obtain a fix and get the data

4.3.9 PmodSWT

Four slides switches for up to 16x different binary logic inputs.

Print switches out to the terminal

Read the four switches and print out to the terminal, sleep for 0.5s and repeat.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2020 RS Components Ltd
# SPDX-License-Identifier: MIT License

"""
Read switch status
"""

from DesignSpark.Pmod.HAT import createPmod
import time

if __name__ == '__main__':

    SWT = createPmod('SWT','JAA')
    time.sleep(0.1)

    try:
        while True:
            print(SWT.GetStatus(1),SWT.GetStatus(2),SWT.GetStatus(3),SWT.GetStatus(4))
            time.sleep(0.5)
    except KeyboardInterrupt:
        pass
    finally:
```

(continues on next page)

(continued from previous page)

```
        SWT.cleanup()
```

Requirements

- PmodSWT module connected to port JAA

4.3.10 PmodLS1

A line follower robot interface system board.

Print received signals from optical sensor out to the terminal

Print the signals from multiple optical sensors out to the terminal, sleep for 0.5s and repeat.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2020 RS Components Ltd
# SPDX-License-Identifier: MIT License

"""
Read Sensor status
"""

from DesignSpark.Pmod.HAT import createPmod
import time

if __name__ == '__main__':

    LS1 = createPmod('LS1','JAA')
    time.sleep(0.1)

    try:
        while True:
            print(LS1.GetAllStatus())
            time.sleep(0.5)
    except KeyboardInterrupt:
        pass
    finally:
        LS1.cleanup()
```

Requirements

- PmodLS1 module connected to port JAA

4.3.11 PmodKYPD

A 16-button keypad.

Print character out to the terminal

Print the key presses out to the terminal, sleep for 0.5s and repeat.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2020 RS Components Ltd
# SPDX-License-Identifier: MIT License

"""
Print keypad presses
"""

from DesignSpark.Pmod.HAT import createPmod
import time

if __name__ == '__main__':

    KYPD = createPmod('KYPD','JA')
    time.sleep(0.1)

    try:
        # set default key map
        KYPD.setKeyMapDefault()

        # set User Key Map
        # keyMap=[['A','B','C','D'],['E','F','G','H'],['I','J','K','L'],['M','N','O',
↪'P']]
        # KYPD.setKeyMap(keyMap)

        # get keyMap
        print(KYPD.getKeyMap())
        while True:
            # print(KYPD.getColRow())
            print(KYPD.getKey())
            time.sleep(0.5)
    except KeyboardInterrupt:
        pass
    finally:
        KYPD.cleanup()
```

Requirements

- PmodKYPD module connected to port JA

4.4 Advanced Examples

4.4.1 OLEDrgb

Organic RGB LED module with a 96×64 pixel display capable of 16-bit color resolution.

Analogue clock

Display an analogue clock face with date and time.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2014-17 Richard Hull and contributors
# See LICENSE.rst for details.
# PYTHON_ARGCOMPLETE_OK

"""
An analog clockface with date & time.

Ported from:
https://gist.github.com/TheRayTracer/dd12c498e3ecb9b8b47f#file-clock-py
"""

import math
import time
import datetime

from DesignSpark.Pmod.HAT import createPmod
from luma.core.render import canvas

def posn(angle, arm_length):
    dx = int(math.cos(math.radians(angle)) * arm_length)
    dy = int(math.sin(math.radians(angle)) * arm_length)
    return (dx, dy)

def main():
    today_last_time = "Unknown"
    while True:
        now = datetime.datetime.now()
        today_date = now.strftime("%d %b %y")
        today_time = now.strftime("%H:%M:%S")
        if today_time != today_last_time:
            today_last_time = today_time
            with canvas(device) as draw:
                now = datetime.datetime.now()
                today_date = now.strftime("%d %b %y")

                margin = 4

                cx = 30
                cy = min(device.height, 64) / 2

                left = cx - cy
                right = cx + cy

                hrs_angle = 270 + (30 * (now.hour + (now.minute / 60.0)))
                hrs = posn(hrs_angle, cy - margin - 7)

                min_angle = 270 + (6 * now.minute)
                mins = posn(min_angle, cy - margin - 2)

                sec_angle = 270 + (6 * now.second)
                secs = posn(sec_angle, cy - margin - 2)
```

(continues on next page)

(continued from previous page)

```
                draw.ellipse((left + margin, margin, right - margin, min(device.
↪height, 64) - margin), outline="white")
                draw.line((cx, cy, cx + hrs[0], cy + hrs[1]), fill="white")
                draw.line((cx, cy, cx + mins[0], cy + mins[1]), fill="white")
                draw.line((cx, cy, cx + secs[0], cy + secs[1]), fill="red")
                draw.ellipse((cx - 2, cy - 2, cx + 2, cy + 2), fill="white", outline=
↪"white")
                draw.text((2 * (cx + margin), cy - 8), today_date, fill="yellow")
                draw.text((2 * (cx + margin), cy), today_time, fill="yellow")

        time.sleep(0.1)

if __name__ == "__main__":
    oled = createPmod('OLEDrgb','JA')
    try:

        device = oled.getDevice()
        main()
    except KeyboardInterrupt:
        pass
    finally:
        oled.cleanup()
```

Game of Life

Conway's Game of Life.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2014-17 Richard Hull and contributors
# See LICENSE.rst for details.
# PYTHON_ARGCOMPLETE_OK

"""
Conway's game of life.

Adapted from:
http://codereview.stackexchange.com/a/108121
"""

import time
from random import randint

from DesignSpark.Pmod.HAT import createPmod
from luma.core.render import canvas

def neighbors(cell):
    x, y = cell
    yield x - 1, y - 1
    yield x, y - 1
    yield x + 1, y - 1
    yield x - 1, y
    yield x + 1, y
    yield x - 1, y + 1
    yield x, y + 1
```

(continues on next page)

(continued from previous page)

```
    yield x + 1, y + 1

def iterate(board):
    new_board = set([])
    candidates = board.union(set(n for cell in board for n in neighbors(cell)))
    for cell in candidates:
        count = sum((n in board) for n in neighbors(cell))
        if count == 3 or (count == 2 and cell in board):
            new_board.add(cell)
    return new_board

def main():
    scale = 3
    cols = device.width // scale
    rows = device.height // scale
    initial_population = int(cols * rows * 0.33)

    while True:
        board = set((randint(0, cols), randint(0, rows)) for _ in range(initial_
↪population))

        for i in range(500):
            with canvas(device, dither=True) as draw:
                for x, y in board:
                    left = x * scale
                    top = y * scale
                    if scale == 1:
                        draw.point((left, top), fill="white")
                    else:
                        right = left + scale
                        bottom = top + scale
                        draw.rectangle((left, top, right, bottom), fill="white",␣
↪outline="black")

                if i == 0:
                    w, h = draw.textsize("Game of Life")
                    left = (device.width - w) // 2
                    top = (device.height - h) // 2
                    draw.rectangle((left - 1, top, left + w + 1, top + h), fill="black
↪", outline="white")
                    draw.text((left + 1, top), text="Game of Life", fill="white")

            if i == 0:
                time.sleep(3)

            board = iterate(board)


if __name__ == "__main__":
    try:
        oled = createPmod('OLEDrgb','JA')
        device = oled.getDevice()
        main()
    except KeyboardInterrupt:
        pass
```

Requirements

- PmodOLEDrgb connected to port JA

4.5 API

4.5.1 DesignSpark.Pmod.AD1

Interface for PmodAD1 module (AD7476A).

Note: Only a single channel (A1) is supported at present due to the way that SPI is configured.

class DesignSpark.Pmod.AD1.**PmodAD1**(*DSPMod6*)

cleanup()

readA1()

readA1Volts()

4.5.2 DesignSpark.Pmod.Error

exception DesignSpark.Pmod.Error.**Error**
Bases: exceptions.Exception

Base class for exceptions in this library.

exception DesignSpark.Pmod.Error.**incorrectModuleName**
Bases: *DesignSpark.Pmod.Error.Error*

Exception raised when the module name given does not exist in the module map.

exception DesignSpark.Pmod.Error.**incorrectPortName**
Bases: *DesignSpark.Pmod.Error.Error*

Exception raised when the port name given does not exist in the port map.

exception DesignSpark.Pmod.Error.**portCapabilityConflict**
Bases: *DesignSpark.Pmod.Error.Error*

Exception raised when the port is already using shared GPIO pins.

exception DesignSpark.Pmod.Error.**portCapabilitySupport**
Bases: *DesignSpark.Pmod.Error.Error*

Exception raised when a port does not support the module type.

exception DesignSpark.Pmod.Error.**portInUse**
Bases: *DesignSpark.Pmod.Error.Error*

Exception raised when the port is already assigned.

4.5.3 DesignSpark.Pmod.HAT

Manages Pmod HAT port resources, enforcing correct usage and avoiding conflicts.

class DesignSpark.Pmod.HAT.**DSPMod12**(*_portName*)

inUse()

setUseModule(*moduleName*)

class DesignSpark.Pmod.HAT.**DSPMod6**(*_portName*)

inUse()

setUseModule(*moduleName*)

DesignSpark.Pmod.HAT.**createPmod**(*moduleName*, *portName*)

4.5.4 DesignSpark.Pmod.HB3

Interface for PmodHB3 module.

class DesignSpark.Pmod.HB3.**PmodHB3**(*DSPMod6*)

changeFrequency(*freq*)

cleanup()

forward(*duty*)

reverse(*duty*)

stop()

4.5.5 DesignSpark.Pmod.ISNS20

Interface for PmodISNS20 module (ADC7476 + Allegro ACS722).

class DesignSpark.Pmod.ISNS20.**PmodISNS20**(*DSPMod6*)

cleanup()

readAmps()

readMilliAmps()

4.5.6 DesignSpark.Pmod.MIC3

Interface for PmodMIC3 (ADCS7476 + Knowles Acoustics SPA2410LR5H-B).

class DesignSpark.Pmod.MIC3.**PmodMIC3**(*DSPMod6*)

MIC3_NO_BITS = 12

cleanup()

dReference = 3.3

readIntegerValue()

readPhysicalValue()

4.5.7 DesignSpark.Pmod.OLEDrgb

Interface for PmodOLEDrgb module (ssd1331).

Note: Depends on luma.oled and luma.core.

class DesignSpark.Pmod.OLEDrgb.**PmodOLEDrgb**(*DSPMod12*)

cleanup()

getDevice()

powerOff()

powerOn()

4.5.8 DesignSpark.Pmod.TC1

Interface for PmodTC1 module (MAX31855).

class DesignSpark.Pmod.TC1.**PmodTC1**(*DSPMod6*)

cleanup()

readCelcius()

readError()

readFarenheit()

readInternal()

4.5.9 DesignSpark.Pmod.ACL2

Interface for Pmod ACL2 module.

class DesignSpark.Pmod.ACL2.**PmodACL2**(*DSPMod12*)

cleanup()

getDeviceID()

getPowerMode()

getRawXYZT()

getSensorStatus()

getTemperature()

getX()

getXYZT()

getY()

getZ()

setOutputRate(*outRate*)

setPowerMode(*PowerMode*)

setRange(*newRange*)

softwareReset()

4.5.10 DesignSpark.Pmod.GPS

Interface for PmodGPS

class DesignSpark.Pmod.GPS.**MicropyGPS**(*local_offset=0*, *location_formatting='ddm'*)

GPS NMEA Sentence Parser. Creates object that stores all relevant GPS data and statistics. Parses sentences one character at a time using update().

SENTENCE_LIMIT = 90

compass_direction()

Determine a cardinal or inter-cardinal direction based on current course. :return: string

date_string(*formatting='s_mdy'*, *century='20'*)

Creates a readable string of the current date. Can select between long format: Januray 1st, 2014 or two short formats: 11/01/2014 (MM/DD/YYYY) 01/11/2014 (DD/MM/YYYY) :param formatting: string 's_mdy', 's_dmy', or 'long' :param century: int delineating the century the GPS data is from (19 for 19XX, 20 for 20XX) :return: date_string string with long or short format date

gpgga()

Parse Global Positioning System Fix Data (GGA) Sentence. Updates UTC timestamp, latitude, longitude, fix status, satellites in use, Horizontal Dilution of Precision (HDOP), altitude, geoid height and fix status

gpgll()

Parse Geographic Latitude and Longitude (GLL)Sentence. Updates UTC timestamp, latitude, longitude, and fix status

gpgsa()

Parse GNSS DOP and Active Satellites (GSA) sentence. Updates GPS fix type, list of satellites used in fix calculation, Position Dilution of Precision (PDOP), Horizontal Dilution of Precision (HDOP), Vertical Dilution of Precision, and fix status

gpgsv()

Parse Satellites in View (GSV) sentence. Updates number of SV Sentences,the number of the last SV sentence parsed, and data on each satellite present in the sentence

gprmc()

Parse Recommended Minimum Specific GPS/Transit data (RMC)Sentence. Updates UTC timestamp, latitude, longitude, Course, Speed, Date, and fix status

gpvtg()

Parse Track Made Good and Ground Speed (VTG) Sentence. Updates speed and course

latitude

Format Latitude Data Correctly

latitude_string()

Create a readable string of the current latitude data :return: string

longitude
Format Longitude Data Correctly

longitude_string()
Create a readable string of the current longitude data :return: string

new_fix_time()
Updates a high resolution counter with current time when fix is updated. Currently only triggered from GGA, GSA and RMC sentences

new_sentence()
Adjust Object Flags in Preparation for a New Sentence

satellite_data_updated()
Checks if the all the GSV sentences in a group have been read, making satellite data complete :return: boolean

satellites_visible()
Returns a list of of the satellite PRNs currently visible to the receiver :return: list

speed_string(*unit='kph'*)
Creates a readable string of the current speed data in one of three units :param unit: string of 'kph','mph, or 'knot' :return:

start_logging(*target_file*, *mode='append'*)
Create GPS data log object

stop_logging()
Closes the log file handler and disables further logging

supported_sentences = {'GLGGA': <function gpgga>, 'GLGLL': <function gpgll>, 'GLGSA':

time_since_fix()
Returns number of millisecond since the last sentence with a valid fix was parsed. Returns 0 if no fix has been found

update(*new_char*)
Process a new input char and updates GPS object if necessary based on special characters ('$', ',', '*') Function builds a list of received string that are validate by CRC prior to parsing by the appropriate sentence function. Returns sentence type on successful parse, None otherwise

write_log(*log_string*)
Attempts to write the last valid NMEA sentence character to the active file handler

class DesignSpark.Pmod.GPS.**PmodGPS**(*DSPMod6*)

cleanup()

getAltitude()

getDate()

getGPSLine()

getGPSPosData()

getHeading()

getLatitude()

getLongitude()

getNumSats()

getPDOP()

getSatelliteData()

getSatelliteInfo()

getSpeedKM()

getTime()

getVisibleSatellite()

gpsUpdate()

isFixed()

4.5.11 DesignSpark.Pmod.SWT

Interface for Pmod_SWT

class DesignSpark.Pmod.SWT.**PmodSWT**(*DSPmod6*)

AllOff()

AllOn()

GetStatus(*SW*)

GetSwitchPin(*SW*)

cleanup()

4.5.12 DesignSpark.Pmod.LS1

Interface for Pmod_LS1

class DesignSpark.Pmod.LS1.**PmodLS1**(*DSPmod6*)

GetAllStatus()

GetSensorPin(*Sensor*)

GetStatus(*SNS*)

cleanup()

4.5.13 DesignSpark.Pmod.KYPD

Interface for Pmod_KYPD

class DesignSpark.Pmod.KYPD.**PmodKYPD**(*DSPmod12*)

cleanup()

getColRow()

getKey()

getKeyMap()

setKeyMap(*UsrKeyMap*)

setKeyMapDefault()

CHAPTER 5

Change log

Version	Description	Date
0.3.0	• Added ACL2, GPS, LS1, SWT & KYPD Pmods	26/02/20
0.2.0	• Tidied up names, added documentation and examples	28/11/17
0.1.0	• Initial version	26/11/17

CHAPTER 6

The MIT License (MIT)

Python Module Index

d

DesignSpark.Pmod.ACL2, 31
DesignSpark.Pmod.AD1, 29
DesignSpark.Pmod.Error, 29
DesignSpark.Pmod.GPS, 32
DesignSpark.Pmod.HAT, 30
DesignSpark.Pmod.HB3, 30
DesignSpark.Pmod.ISNS20, 30
DesignSpark.Pmod.KYPD, 34
DesignSpark.Pmod.LS1, 34
DesignSpark.Pmod.MIC3, 30
DesignSpark.Pmod.OLEDrgb, 31
DesignSpark.Pmod.SWT, 34
DesignSpark.Pmod.TC1, 31

Index

A

AllOff() (*DesignSpark.Pmod.SWT.PmodSWT method*), 34
AllOn() (*DesignSpark.Pmod.SWT.PmodSWT method*), 34

C

changeFrequency() (*DesignSpark.Pmod.HB3.PmodHB3 method*), 30
cleanup() (*DesignSpark.Pmod.ACL2.PmodACL2 method*), 31
cleanup() (*DesignSpark.Pmod.AD1.PmodAD1 method*), 29
cleanup() (*DesignSpark.Pmod.GPS.PmodGPS method*), 33
cleanup() (*DesignSpark.Pmod.HB3.PmodHB3 method*), 30
cleanup() (*DesignSpark.Pmod.ISNS20.PmodISNS20 method*), 30
cleanup() (*DesignSpark.Pmod.KYPD.PmodKYPD method*), 34
cleanup() (*DesignSpark.Pmod.LS1.PmodLS1 method*), 34
cleanup() (*DesignSpark.Pmod.MIC3.PmodMIC3 method*), 30
cleanup() (*DesignSpark.Pmod.OLEDrgb.PmodOLEDrgb method*), 31
cleanup() (*DesignSpark.Pmod.SWT.PmodSWT method*), 34
cleanup() (*DesignSpark.Pmod.TC1.PmodTC1 method*), 31
compass_direction() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 32
createPmod() (*in module DesignSpark.Pmod.HAT*), 30

D

date_string() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 32
DesignSpark.Pmod.ACL2 (*module*), 31
DesignSpark.Pmod.AD1 (*module*), 29
DesignSpark.Pmod.Error (*module*), 29
DesignSpark.Pmod.GPS (*module*), 32
DesignSpark.Pmod.HAT (*module*), 30
DesignSpark.Pmod.HB3 (*module*), 30
DesignSpark.Pmod.ISNS20 (*module*), 30
DesignSpark.Pmod.KYPD (*module*), 34
DesignSpark.Pmod.LS1 (*module*), 34
DesignSpark.Pmod.MIC3 (*module*), 30
DesignSpark.Pmod.OLEDrgb (*module*), 31
DesignSpark.Pmod.SWT (*module*), 34
DesignSpark.Pmod.TC1 (*module*), 31
dReference (*DesignSpark.Pmod.MIC3.PmodMIC3 attribute*), 30
DSPMod12 (*class in DesignSpark.Pmod.HAT*), 30
DSPMod6 (*class in DesignSpark.Pmod.HAT*), 30

E

Error, 29

F

forward() (*DesignSpark.Pmod.HB3.PmodHB3 method*), 30

G

GetAllStatus() (*DesignSpark.Pmod.LS1.PmodLS1 method*), 34
getAltitude() (*DesignSpark.Pmod.GPS.PmodGPS method*), 33
getColRow() (*DesignSpark.Pmod.KYPD.PmodKYPD method*), 34
getDate() (*DesignSpark.Pmod.GPS.PmodGPS method*), 33
getDevice() (*DesignSpark.Pmod.OLEDrgb.PmodOLEDrgb method*), 31

getDeviceID() (*DesignSpark.Pmod.ACL2.PmodACL2 method*), 31
getGPSLine() (*DesignSpark.Pmod.GPS.PmodGPS method*), 33
getGPSPosData() (*DesignSpark.Pmod.GPS.PmodGPS method*), 33
getHeading() (*DesignSpark.Pmod.GPS.PmodGPS method*), 33
getKey() (*DesignSpark.Pmod.KYPD.PmodKYPD method*), 34
getKeyMap() (*DesignSpark.Pmod.KYPD.PmodKYPD method*), 34
getLatitude() (*DesignSpark.Pmod.GPS.PmodGPS method*), 33
getLongitude() (*DesignSpark.Pmod.GPS.PmodGPS method*), 33
getNumSats() (*DesignSpark.Pmod.GPS.PmodGPS method*), 33
getPDOP() (*DesignSpark.Pmod.GPS.PmodGPS method*), 33
getPowerMode() (*DesignSpark.Pmod.ACL2.PmodACL2 method*), 31
getRawXYZT() (*DesignSpark.Pmod.ACL2.PmodACL2 method*), 31
getSatelliteData() (*DesignSpark.Pmod.GPS.PmodGPS method*), 34
getSatelliteInfo() (*DesignSpark.Pmod.GPS.PmodGPS method*), 34
GetSensorPin() (*DesignSpark.Pmod.LS1.PmodLS1 method*), 34
getSensorStatus() (*DesignSpark.Pmod.ACL2.PmodACL2 method*), 31
getSpeedKM() (*DesignSpark.Pmod.GPS.PmodGPS method*), 34
GetStatus() (*DesignSpark.Pmod.LS1.PmodLS1 method*), 34
GetStatus() (*DesignSpark.Pmod.SWT.PmodSWT method*), 34
GetSwitchPin() (*DesignSpark.Pmod.SWT.PmodSWT method*), 34
getTemperature() (*DesignSpark.Pmod.ACL2.PmodACL2 method*), 31
getTime() (*DesignSpark.Pmod.GPS.PmodGPS method*), 34
getVisibleSatellite() (*DesignSpark.Pmod.GPS.PmodGPS method*), 34
getX() (*DesignSpark.Pmod.ACL2.PmodACL2 method*), 31
getXYZT() (*DesignSpark.Pmod.ACL2.PmodACL2 method*), 31
getY() (*DesignSpark.Pmod.ACL2.PmodACL2 method*), 31
getZ() (*DesignSpark.Pmod.ACL2.PmodACL2 method*), 32
gpgga() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 32
gpgll() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 32
gpgsa() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 32
gpgsv() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 32
gprmc() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 32
gpsUpdate() (*DesignSpark.Pmod.GPS.PmodGPS method*), 34
gpvtg() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 32

I

incorrectModuleName, 29
incorrectPortName, 29
inUse() (*DesignSpark.Pmod.HAT.DSPMod12 method*), 30
inUse() (*DesignSpark.Pmod.HAT.DSPMod6 method*), 30
isFixed() (*DesignSpark.Pmod.GPS.PmodGPS method*), 34

L

latitude (*DesignSpark.Pmod.GPS.MicropyGPS attribute*), 32
latitude_string() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 32
longitude (*DesignSpark.Pmod.GPS.MicropyGPS attribute*), 32
longitude_string() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 33

M

MIC3_NO_BITS (*DesignSpark.Pmod.MIC3.PmodMIC3 attribute*), 30
MicropyGPS (*class in DesignSpark.Pmod.GPS*), 32

N

new_fix_time() (*De-*

signSpark.Pmod.GPS.MicropyGPS method), 33
new_sentence() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 33

P

PmodACL2 (*class in DesignSpark.Pmod.ACL2*), 31
PmodAD1 (*class in DesignSpark.Pmod.AD1*), 29
PmodGPS (*class in DesignSpark.Pmod.GPS*), 33
PmodHB3 (*class in DesignSpark.Pmod.HB3*), 30
PmodISNS20 (*class in DesignSpark.Pmod.ISNS20*), 30
PmodKYPD (*class in DesignSpark.Pmod.KYPD*), 34
PmodLS1 (*class in DesignSpark.Pmod.LS1*), 34
PmodMIC3 (*class in DesignSpark.Pmod.MIC3*), 30
PmodOLEDrgb (*class in DesignSpark.Pmod.OLEDrgb*), 31
PmodSWT (*class in DesignSpark.Pmod.SWT*), 34
PmodTC1 (*class in DesignSpark.Pmod.TC1*), 31
portCapabilityConflict, 29
portCapabilitySupport, 29
portInUse, 29
powerOff() (*DesignSpark.Pmod.OLEDrgb.PmodOLEDrgb method*), 31
powerOn() (*DesignSpark.Pmod.OLEDrgb.PmodOLEDrgb method*), 31

R

readA1() (*DesignSpark.Pmod.AD1.PmodAD1 method*), 29
readA1Volts() (*DesignSpark.Pmod.AD1.PmodAD1 method*), 29
readAmps() (*DesignSpark.Pmod.ISNS20.PmodISNS20 method*), 30
readCelcius() (*DesignSpark.Pmod.TC1.PmodTC1 method*), 31
readError() (*DesignSpark.Pmod.TC1.PmodTC1 method*), 31
readFarenheit() (*DesignSpark.Pmod.TC1.PmodTC1 method*), 31
readIntegerValue() (*DesignSpark.Pmod.MIC3.PmodMIC3 method*), 30
readInternal() (*DesignSpark.Pmod.TC1.PmodTC1 method*), 31
readMilliAmps() (*DesignSpark.Pmod.ISNS20.PmodISNS20 method*), 30
readPhysicalValue() (*DesignSpark.Pmod.MIC3.PmodMIC3 method*), 31
reverse() (*DesignSpark.Pmod.HB3.PmodHB3 method*), 30

S

satellite_data_updated() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 33
satellites_visible() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 33
SENTENCE_LIMIT (*DesignSpark.Pmod.GPS.MicropyGPS attribute*), 32
setKeyMap() (*DesignSpark.Pmod.KYPD.PmodKYPD method*), 34
setKeyMapDefault() (*DesignSpark.Pmod.KYPD.PmodKYPD method*), 35
setOutputRate() (*DesignSpark.Pmod.ACL2.PmodACL2 method*), 32
setPowerMode() (*DesignSpark.Pmod.ACL2.PmodACL2 method*), 32
setRange() (*DesignSpark.Pmod.ACL2.PmodACL2 method*), 32
setUseModule() (*DesignSpark.Pmod.HAT.DSPMod12 method*), 30
setUseModule() (*DesignSpark.Pmod.HAT.DSPMod6 method*), 30
softwareReset() (*DesignSpark.Pmod.ACL2.PmodACL2 method*), 32
speed_string() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 33
start_logging() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 33
stop() (*DesignSpark.Pmod.HB3.PmodHB3 method*), 30
stop_logging() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 33
supported_sentences (*DesignSpark.Pmod.GPS.MicropyGPS attribute*), 33

T

time_since_fix() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 33

U

update() (*DesignSpark.Pmod.GPS.MicropyGPS*

method), 33

W

write_log() (*DesignSpark.Pmod.GPS.MicropyGPS method*), 33