```
classdef ads1115 < handle & matlab.mixin.CustomDisplay
    %ADS1115 Analog-to-Digital converter.
    %
    % adc = ads1115(rpi, bus) creates a ADS1115 ADC object attached to the
    % specified I2C bus. The first parameter, rpi, is a raspi object. The
    % I2C address of the ADS1115 ADC defaults to '0x48'.
    %
    % adc = ads1115(rpi, bus, address) creates a ADS1115 ADC object
    % attached to the specified I2C bus and I2C address. Use this form if
    % you used the ADDR pin to change the I2C address of the ADS1115 from
    % the default '0x48' to something else.
    %
    % readVoltage(adc, AINp) reads the single-ended voltage measurement
    % from AINp input port.
    %
    % readVoltage(adc, AINp, AINn) reads the differential voltage
    % measurement between AINp and AINn input ports.
    %
    % The OperatingMode property of the ADS1115 ADC object determines power
    % consumption, speed and accuracy. The default OperatingMode is
    % 'single-shot' meaning that the ADS1115 performs a single analog to
    % digital conversion upon request and goes to power save mode. In
    % continuous mode, the device performs continuous conversions.
    %
    % The SamplesPerSecond property sets the conversion rate.
    %
    % The VoltageScale property of the ADS1115 ADC object determines the
    % setting of the Programmable Gain Amplifier (PGA) value applied before
    % analog to digital conversion. See table below to correlate the
    % input voltage scale with the PGA value:
    %
    % VoltageScale | PGA Value
    % -------------------------
    %    6.144  |    2/3
    %    4.096  |    1
    %    2.048  |    2
    %    1.024  |    4
    %    0.512  |    8
    %    0.256  |    16
    %
    % <a href="http://www.ti.com/lit/gpn/ads1115">Device Datasheet</a>
    %
    % NOTE: Do not apply voltages excedding VDD+0.3V to any input pin.

    % Copyright 2014 The MathWorks, Inc.

    properties (SetAccess = private, GetAccess = public)
        Address = bin2dec('1001000') % Default address 0x48
    end

    properties (Access = public)
```

```
        OperatingMode
        VoltageScale
        SamplesPerSecond
    end

    properties (Access = private)
        i2cObj
        PGAbits
        AINp
        AINn
        NumInputs = 4
        ConfigReg
    end

    properties (Constant, Hidden)
        AvailableSamplesPerSecond = [8, 16, 32, 64, 128, 250, 475, 860]
        AvailableVoltageScale = [6.144, 4.096, 2.048, 1.024, 0.512, 0.256]
        AvailableOperatingMode = {'single-shot', 'continuous'}
    end

    properties (Constant, Access = private)
        % Register addresses
        CONVERSION_REG = 0
        CONFIG_REG     = 1
        LOTHRESH_REG   = 2
        HITHRESH_REG   = 3

        % Config register bit shifts
        CONFIG_OS_SHIFT        = 15
        CONFIG_MUX_SHIFT       = 12
        CONFIG_PGA_SHIFT       = 9
        CONFIG_MODE_SHIFT      = 8
        CONFIG_DR_SHIFT        = 5
        CONFIG_COMP_MODE_SHIFT = 4
        CONFIG_COMP_POL_SHIFT  = 3
        CONFIG_COMP_LAT_SHIFT  = 2
        CONFIG_COMP_QUE_SHIFT  = 0

        % Full scale for ADS1115 is 4.096 volts
        FS_VOLTAGE = 4.096

        % 16-bit ADC result needs to be scaled by this value
        ADC_SCALAR = 2^15 - 1
    end

    methods
        function obj = ads1115(raspiObj, address)
            % Set I2C address if not using default
            if nargin > 1
                obj.Address = address;
            end
            % Set defaults
            obj.SamplesPerSecond = 128;
            obj.OperatingMode = 'single-shot';
            obj.VoltageScale = 4.096;

            % Initialize config register value
            obj.ConfigReg = 0;

            % Create an i2cdev object to talk to ADS1115
            obj.i2cObj = i2cdev(raspiObj, obj.Address);
        end

        function voltage = readVoltage(obj, AINp, AINn)
            % voltage = readVoltage(obj, AINp) reads the single-ended input
            % voltage value at channe AINp.
            %
            % voltage = readVoltage(obj, AINp, AINn) reads the input
            % voltage value that is the difference between AINp and AINn.
            validateattributes(AINp, {'numeric'}, ...
                {'scalar', '>=', 0, '<=', obj.NumInputs-1}, '', 'AINp');
            if nargin > 2
                validateattributes(AINn, {'numeric'}, ...
                    {'scalar', '>=', 0, '<=', obj.NumInputs-1}, '', 'AINn');
            else
                AINn = -1;
            end

            % Configure ADC and read requested conversion value
            configReg = getConfigReg(obj, AINp, AINn);
            if isequal(obj.OperatingMode, 'single-shot') || ...
                    (configReg ~= obj.ConfigReg)
                obj.ConfigReg = configReg;
                configureDevice(obj);
            end

            % Each I2C transaction with raspi object takes about 5ms. If
            % conversion time is greater than this we must wait
            if isequal(obj.OperatingMode, 'single-shot') && ...
                    (1/obj.SamplesPerSecond > 0.005)
                pause(1/obj.SamplesPerSecond);
            end

            % Read raw ADC conversion value and convert to voltage
            data = readRegister(obj.i2cObj, obj.CONVERSION_REG, 'int16');
            voltage = double(swapbytes(data)) * (obj.VoltageScale) / obj.ADC_SCALAR;
        end
    end

    methods
        function set.Address(obj, value)
```

```
            if isnumeric(value)
                validateattributes(value, {'numeric'}, ...
                    {'scalar', 'nonnegative'}, '', 'Address');
            else
                validateattributes(value, {'char'}, ...
                    {'nonempty'}, '', 'Address');
                value = obj.hex2dec(value);
            end
            if (value < obj.hex2dec('0x48')) || (value > obj.hex2dec('0x51'))
                error('raspi:ads1115:InvalidI2CAddress', ...
                    'Invalid I2C address. I2C address must be one of the following: 0x48,↙
0x49, 0x50, 0x51');
            end
            obj.Address = value;
        end

        function set.SamplesPerSecond(obj, value)
            validateattributes(value, {'numeric'}, ...
                {'scalar', 'nonnan', 'finite'}, '', 'SamplesPerSecond');
            if ~ismember(value, obj.AvailableSamplesPerSecond)
                error('raspi:ads1115:InvalidSamplesPerSecond', ...
                    'SamplesPerSecond must be one of the following: %d', ...
                    obj.AvailableSamplesPerSecond);
            end
            obj.SamplesPerSecond = value;
        end

        function set.VoltageScale(obj, value)
            validateattributes(value, {'numeric'}, ...
                {'scalar', 'nonnan', 'finite'}, '', 'VoltageScale');
            if ~ismember(value, obj.AvailableVoltageScale)
                error('raspi:ads1115:InvalidVoltageScale', ...
                    'VoltageScale must be one of the following: %d', ...
                    obj.AvailableVoltageScale);
            end
            obj.VoltageScale = value;
            switch obj.VoltageScale
                case 6.144,
                    obj.PGAbits = 0; %#ok<*MCSUP>
                case 4.096,
                    obj.PGAbits = 1;
                case 2.048,
                    obj.PGAbits = 2;
                case 1.024,
                    obj.PGAbits = 3;
                case 0.512,
                    obj.PGAbits = 4;
                case 0.256
                    obj.PGAbits = 5;
            end
        end
        function set.OperatingMode(obj, value)
            value = validatestring(value, obj.AvailableOperatingMode);
            obj.OperatingMode = value;
        end
    end

    methods (Access = protected)
        function displayScalarObject(obj)
            header = getHeader(obj);
            disp(header);

            % Display main options
            fprintf('                Address: %-15s\n', ['0x' dec2hex(obj.Address)]);
            fprintf('          OperatingMode: %-15s (''single-shot'' or ''continuous'')↙
\n', ...
                obj.OperatingMode);
            fprintf('       SamplesPerSecond: %-15d (8, 16, 32, 64, 128, 250, 475, or↙
860)\n', ...
                obj.SamplesPerSecond);
            fprintf('           VoltageScale: %-15.3f (6.144, 4.096, 2.048, 1.024, 0.512,↙
or 0.256)\n', ...
                obj.VoltageScale);
            fprintf('\n');

            % Allow for the possibility of a footer.
            footer = getFooter(obj);
            if ~isempty(footer)
                disp(footer);
            end
        end

        function configReg = getConfigReg(obj, AINp, AINn)
            % Disable comparator
            configReg = bitshift(bin2dec('11'), obj.CONFIG_COMP_QUE_SHIFT);

            % Set samples per second bits DR[2:0]
            switch obj.SamplesPerSecond
                case 8
                    DRbits = 0;
                case 16
                    DRbits = 1;
                case 32
                    DRbits = 2;
                case 64
                    DRbits = 3;
                case 128
                    DRbits = 4;
                case 250
                    DRbits = 5;
                case 475
```

```
                    DRbits = 6;
                case 860
                    DRbits = 7;
            end
            configReg = bitor(configReg, bitshift(DRbits, obj.CONFIG_DR_SHIFT));

            % Set operating mode bits MODE[8]
            if isequal(obj.OperatingMode, 'single-shot')
                MODEbits = 1;
                configReg = bitor(configReg, bitshift(MODEbits, obj.CONFIG_MODE_SHIFT));
                configReg = bitor(configReg, bitshift(1, obj.CONFIG_OS_SHIFT));
            end

            % Set PGA bits PGA[2:0]
            configReg = bitor(configReg, bitshift(obj.PGAbits, obj.CONFIG_PGA_SHIFT));

            % Set MUX bits MUX[2:0]
            if AINn == -1
                switch AINp
                    case 0,
                        MUXbits = bin2dec('100');
                    case 1,
                        MUXbits = bin2dec('101');
                    case 2,
                        MUXbits = bin2dec('110');
                    case 3,
                        MUXbits = bin2dec('111');
                end
            else
                if (AINp == 0) && (AINn == 1)
                    MUXbits = 0;
                elseif (AINp == 0) && (AINn == 3)
                    MUXbits = 1;
                elseif (AINp == 1) && (AINn == 3)
                    MUXbits = 2;
                elseif (AINp == 2) && (AINn == 3)
                    MUXbits = 3;
                else
                    error('raspi:ads1115:InvalidAIN', ...
                        ['Invalid (AINp, AINn) pair for differential voltage measurement.↙
', ...
                        'Supported (AINp, AINn) values are: (0, 1), (0, 3), (1, 3), (2,↙
3).']);
                end
            end
            configReg = bitor(configReg, bitshift(MUXbits, obj.CONFIG_MUX_SHIFT));
        end
    end

    methods (Access = private)
        function configureDevice(obj)
```

```
            obj.i2cObj.writeRegister(obj.CONFIG_REG, ...
                swapbytes(uint16(obj.ConfigReg)), 'uint16');
        end

        function reg = readConfigReg(obj)
            reg = swapbytes(readRegister(obj.i2cObj, obj.CONFIG_REG, 'uint16'));
        end
    end

    methods (Static)
        function decvalue = hex2dec(hexvalue)
            decvalue = hex2dec(regexprep(hexvalue, '0x', ''));
        end

        function hexvalue = dec2hex(decvalue)
            hexvalue = sprintf('0x%02s', dec2hex(decvalue));
        end
    end
end
```