Mechatronics System Integration (MCTA3203)

Week **11**: Digital Signal Processing (DSP) interfacing

Digital signal processing (DSP) involves in utilizing mathematical algorithms to manipulate and analyze digital signals. These signals are representation of information in a form of electrical voltage, current, sound, light or any physical quantities that can be measured. Below are some of the elements in DSP.

- Digital Signals: DSP deals with digital signals, which are discrete-time representations of continuous signals. Digital signals are obtained through processes like sampling and quantization.
- Analog-to-Digital Conversion (ADC): This process involves converting continuous analog signals into digital form, making them suitable for processing by digital systems.
- Digital-to-Analog Conversion (DAC): After processing in the digital domain, signals often need to be converted back to analog form for output or transmission.
- Mathematical Algorithms: DSP relies heavily on mathematical algorithms for operations like filtering, convolution, Fourier analysis, and modulation. These algorithms are implemented using digital techniques.
- Filters: Filters are used to modify or extract specific components of a signal. Common types include low-pass, high-pass, band-pass, and notch filters.
- Fast Fourier Transform (FFT): The FFT is a crucial algorithm in DSP that efficiently computes the frequency content of a signal, allowing for spectral analysis.
- Signal Processing Blocks: These are functional units that perform specific operations on signals. Examples include amplifiers, mixers, modulators, and demodulators.
- Digital Signal Processors (DSP Processors): Specialized hardware designed to efficiently execute DSP algorithms. These processors are optimized for fast and efficient signal processing tasks.

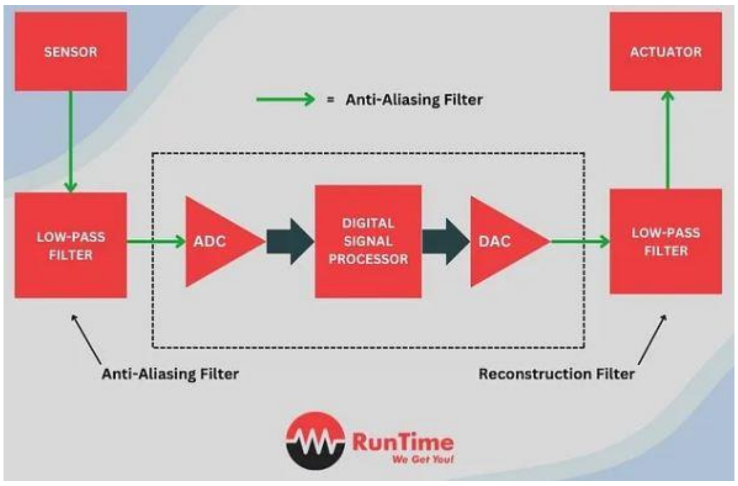

Fig.1 Digital Signal Processing

https://medium.com/@lanceharvieruntime/digital-signal-processing-dsp-in-embedded-systems-68b4649ff441

TASK

Demonstrate the low pass filter (LPF) function using arduino. Analog input may come from sensor, function generator or audio signals. Output must shown in the in a form of waveforms in the serial monitor of the Arduino IDE.

Materials

- Arduino board
- Arduino IDE
- Breadboard and jumper wires
- Analog input (sensors/ function generator etc.)
- Resistor and Capacitor (for LPF and HPF)

Procedures:

1. Connect the sensor to the analog input of the arduino through LPF as shown in Fig. 2 below.

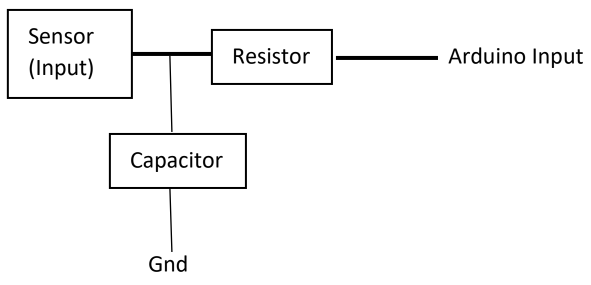

Fig.2

2. The the resistor and capacitor form a low-pass filter circuit which conforms to the cut-off frequency formula, $f_c = 1/2\pi RC$. Choose the value resistor and capacitor based on desired cut-off frequency.
3. Write arduino code to read analog data from sensor and implement a low-pass filter.
4. Upload the code to arduino board.
5. Monitor the output using the serial monitor in IDE.
6. Observe how LPF smoothen the signal and reduces high frequency noise.
7. Discuss the principle of low-pass filter and its application. Consider real-world applications of digital filtering in areas like audio processing or sensor data filtering.

*Repeat the procedures in the Task above and use high-pass filter instead of low-pass filter. Demonstrate and observe the function of a high-pass filter.

Prepared by: Dr. Nadzril Sulaiman

References

1. https://medium.com/@lanceharvieruntime/digital-signal-processing-dsp-in-embedded-systems-68b4649ff441
2. https://www.rs-online.com/designspark/getting-into-digital-signal-processing-a-basic-introduction
3. https://www.youtube.com/watch?v=JXGlq6THyWU

Example:
Below is a simple Arduino code for implementing a low-pass filter. This code reads an analog signal from a sensor, applies a low-pass filter to smooth the signal, and prints both the original and filtered values to the Serial Monitor. This example uses a first-order low-pass filter with an exponential moving average.

```
const int analogPin = A0;
int sensorValue;
float filteredValue = 0.0;
float alpha = 0.1; // Adjust this value for filter strength

void setup() {
  Serial.begin(9600);
}

void loop() {
  // Read analog data from the sensor
  sensorValue = analogRead(analogPin);

  // Apply low-pass filter
  filteredValue = alpha * sensorValue + (1 - alpha) * filteredValue;

  // Print the results
  Serial.print("Original: ");
  Serial.print(sensorValue);
  Serial.print("\tFiltered: ");
  Serial.println(filteredValue);

  delay(100); // Adjust the delay based on your sampling rate
}
```

Explanation:

- **analogPin**: Represents the analog pin to which the sensor is connected.
- **sensorValue**: Holds the raw analog reading from the sensor.
- **filteredValue**: Represents the filtered value after applying the low-pass filter.
- **alpha**: Is a smoothing factor that determines the strength of the low-pass filter. It should be between 0 and 1. Adjust this value based on the desired filtering strength.

In each iteration of the **loop()** function:

1. The code reads the analog value from the sensor.
2. The low-pass filter is applied to smooth the signal using the formula filteredValue=alpha×sensorValue+(1−alpha)×filteredValuefilteredValue=alpha×sensorValue+(1−alpha)×filteredValue.
3. The original and filtered values are printed to the Serial Monitor.
4.

Feel free to adjust the **alpha** value and experiment with different filtering strengths. Also, consider incorporating this filtering mechanism into your specific application or sensor setup.