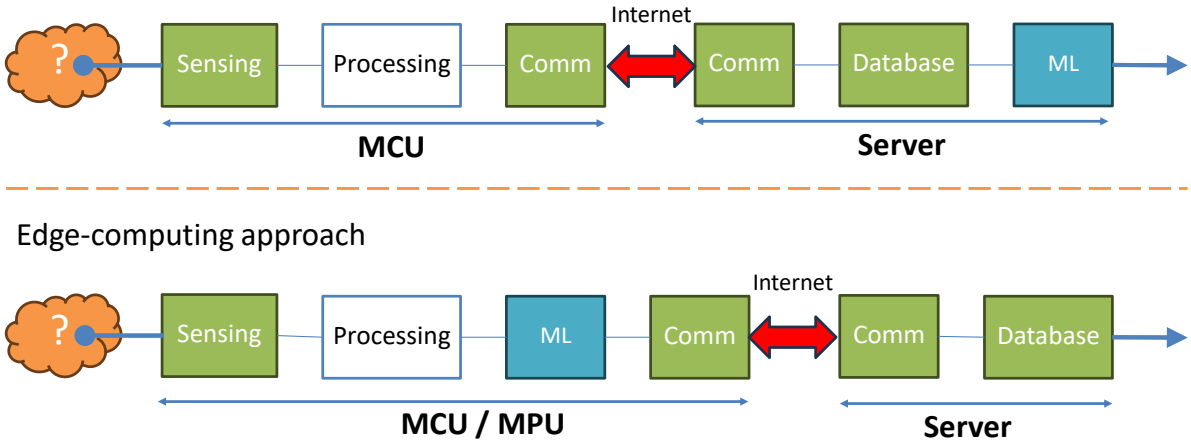
MATLAB Coder workshop
code generation for Raspberry Pi

ผศ.ดร.ศุภชัย วรพจน์พิศุทธิ์
ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์
ม.ธรรมศาสตร์

Learning outcomes

Objectives	Practice
1. Learn MATLAB Coder workflow	1. Use MATLAB Coder app to generate C code
2. Learn Simulink code generation	2. Use Simulink Hardware Support Packages with Embedded Coder
3. Understand sound capture	3. Write C code to use ALSA library
4. Practice sound processing	4. Develop MATLAB for FFT, then generate C code, and build

AIoT architecture



AIoT architecture

Internet of Things (IoT), with the help of some technologies such as 5G, fog/edge computing, and Artificial Intelligence (AI), is driving a significant shift to support emerging applications with specific requirements (e.g., real-time processing of a high volume of data) with greater flexibility and efficiency. In this regard, AI subfields such as data analytics and learning techniques play an important role to support the emerging IoT-related applications. AI integrated or combined with the IoT heralds the era of Artificial Intelligence of Things (AIoT).

In 2018, KPMG published a foresight study on the future of AI including scenarios until 2040. The analysts describe a scenario in detail where a community of things would see each device also contain its own AI that could link autonomously to other AIs to, together, perform tasks intelligently. Value creation would be controlled and executed in real-time using swarm intelligence. Many industries could be transformed with the application of swarm intelligence, including: automotive, cloud, medical, military, research, and technology.

Commonly referred to as IoT cloud, cloud-based IoT is the management and processing of data from IoT devices using cloud computing platforms. Connecting IoT devices to the cloud is essential since that's where data is stored, processed and accessed by various applications and services. Cloud-based AIoT is composed of the following four layers:

- **Device layer:** This includes several types of hardware, including tags, beacons, sensors, cars, production equipment, embedded devices, and health and fitness equipment.
- **Connectivity layer**. This layer comprises fields and cloud gateways consisting of a hardware or software element that links cloud storage to controllers, sensors and other intelligent devices.
- **Cloud layer:** This consists of data processing via an AI engine, data storage, data visualization, analytics and data access via an API.
- **User communication layer:** This layer is made up of web portals + mobile applications.

Audio signal processing with RPi + MATLAB

MATLAB and Raspberry Pi

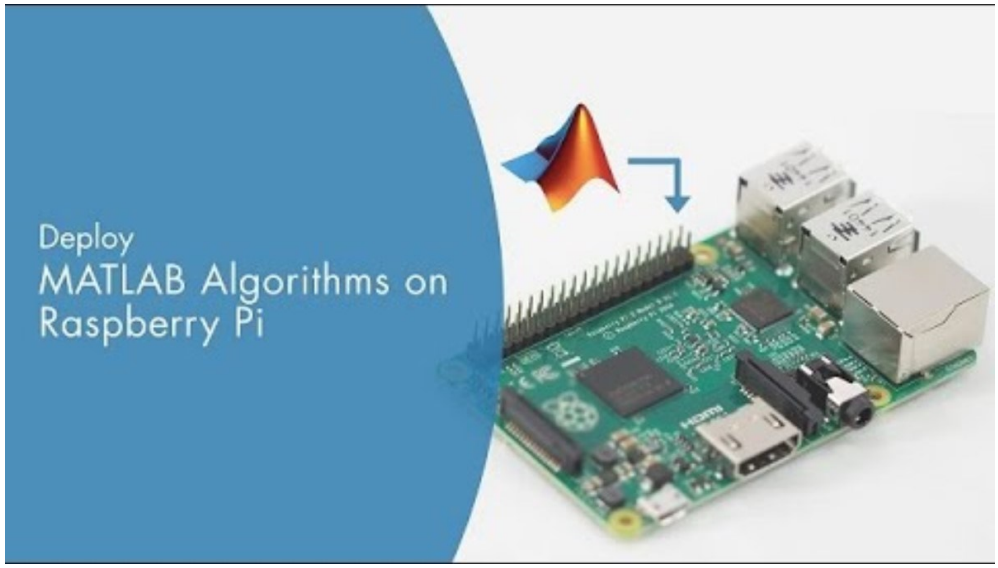
MATLAB Support Package for Raspberry Pi™ Hardware provides two ways of programming Raspberry Pi applications from MATLAB.

- **Interactive communication:** You can remotely communicate with a Raspberry Pi from a desktop installation of MATLAB or through a web browser with MATLAB Online. Acquire data from sensors and imaging devices connected to the Raspberry Pi and then analyze and visualize it in MATLAB.
- **Standalone execution:** With MATLAB Coder, you can develop standalone embedded applications for Raspberry Pi. Use the interactive communication to prototype and develop your MATLAB algorithm, then automatically generate equivalent C code and deploy it to the Raspberry Pi to run as a standalone application.

Using MATLAB Coder™ support for Raspberry Pi™, you can prototype an audio processing application on a Raspberry Pi. This example demonstrates how you can capture live audio feed from a webcam or USB microphone, run the deployed pitch shifting algorithm on the hardware, and play back the modified audio data through the headphone jack of the Raspberry Pi board.

https://www.mathworks.com/hardware-support/raspberry-pi-matlab.html
https://www.youtube.com/watch?v=cwY45Ur0Teo

Algorithm development with RPi + MATLAB



MATLAB code generation

MATLAB Coder generates C and C++ code from MATLAB code for a variety of hardware platforms, from desktop systems to embedded hardware. It supports most of the MATLAB language and a wide range of toolboxes. You can integrate the generated code into your projects as source code, static libraries, or dynamic libraries. The generated code is readable and portable. You can combine it with key parts of your existing C and C++ code and libraries. You can also package the generated code as a MEX-function for use in MATLAB.

Prepare MATLAB code for code generation and generate C/C++ code.

STEP 1: Prepare MATLAB Code for Code Generation

STEP 2: Generate C/C++ Code from MATLAB Code

STEP 3: Test Generated C/C++ Code

STEP 4: Deploy Generated C/C++ Code

When used with Embedded Coder, MATLAB Coder provides code customizations, target-specific optimizations, code traceability, and software-in-the-loop (SIL) and processor-in-the-loop (PIL) verification.

https://www.mathworks.com/help/coder/getting-started-with-matlab-coder.html
https://www.mathworks.com/help/coder/gs/code-generation-guide-generate-deployable-cc-code.html
https://www.youtube.com/watch?v=5lkCx8-zJm0

MATLAB/Simulink code generation

Q1: scope	☐ Application	☐ Function	
Q2: processor	☐ Arduino / Raspberry Pi	☐ Linux board	☐ MCU
Q3: HW support package	☐ Yes	☐ No	
Q4: Special HW	☐ No	☐ On-chip	☐ On-board
Q5: Special SW	☐ No	☐ Protocol	☐ …
Q6: Timing	☐ ≤ 10 Hz	☐ ≤ 100 Hz	☐ > 100 Hz

Simulink	Simulink + custom block	MATLAB coder + dev toolchain	Embedded coder
• Model	• Model • Subsystem • Library	• Application • Function • Library	• Project • Function

MATLAB Coder

MATLAB Coder generates C and C++ code from MATLAB code for a variety of hardware platforms, from desktop systems to embedded hardware. It supports most of the MATLAB language and a wide range of toolboxes. You can integrate the generated code into your projects as source code, static libraries, or dynamic libraries. The generated code is readable and portable. You can combine it with key parts of your existing C and C++ code and libraries.
https://www.mathworks.com/products/matlab-coder.html

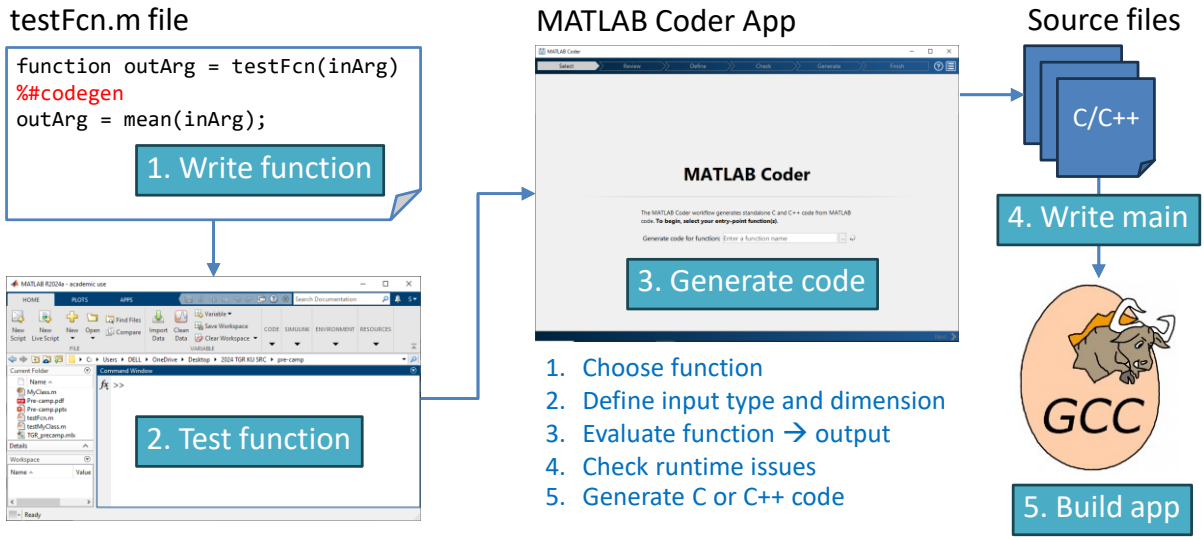
Simulink Coder

Simulink Coder generates and executes C and C++ code from Simulink models, Stateflow charts, and MATLAB functions. The generated source code can be used for real-time and nonreal-time applications, including simulation acceleration, rapid prototyping, and hardware-in-the-loop testing. You can tune and monitor the generated code using Simulink or run and interact with the code outside MATLAB and Simulink.
https://www.mathworks.com/products/simulink-coder.html

Embedded Coder

Embedded Coder generates readable, compact, and fast C and C++ code for embedded processors used in mass production. It extends MATLAB Coder and Simulink Coder with advanced optimizations for precise control of the generated functions, files, and data. Embedded Coder offers built-in support for AUTOSAR, MISRA C, and ASAP2 software standards. Embedded Coder code is portable and can be compiled and executed on any processor.
https://www.mathworks.com/products/embedded-coder.html

MATLAB Coder workflow



MATLAB Coder

MATLAB® Coder™ generates C and C++ code from MATLAB code for a variety of hardware platforms, from desktop systems to embedded hardware. It supports most of the MATLAB language and a wide range of toolboxes. You can integrate the generated code into your projects as source code, static libraries, or dynamic libraries. The generated code is readable and portable. When used with Embedded Coder®, MATLAB Coder provides code customizations, target-specific optimizations, code traceability, and software-in-the-loop (SIL) and processor-in-the-loop (PIL) verification.

The procedure to prepare MATLAB code for code generation and generate C/C++ code.

1. Prepare MATLAB Code for Code Generation
2. Generate C/C++ Code from MATLAB Code
3. Test Generated C/C++ Code
4. Deploy Generated C/C++ Code

MATLAB provides **coder** command to start MATLAB Coder App, or use **codegen** command generate a static C library. Available command options are:

- **coder.screener** to run the Code Generation Readiness Tool
- **coder.config** reate code generation configuration objects
- **coder.ceval** to call custom C functions that replace the unsupported functions.
- **coder.extrinsic** to call the unsupported functions.

https://www.mathworks.com/help/coder/ug/code-generation-workflow.html
https://www.mathworks.com/help/coder/gs/generating-c-code-from-matlab-code-using-the-matlab-coder-project-interface.html

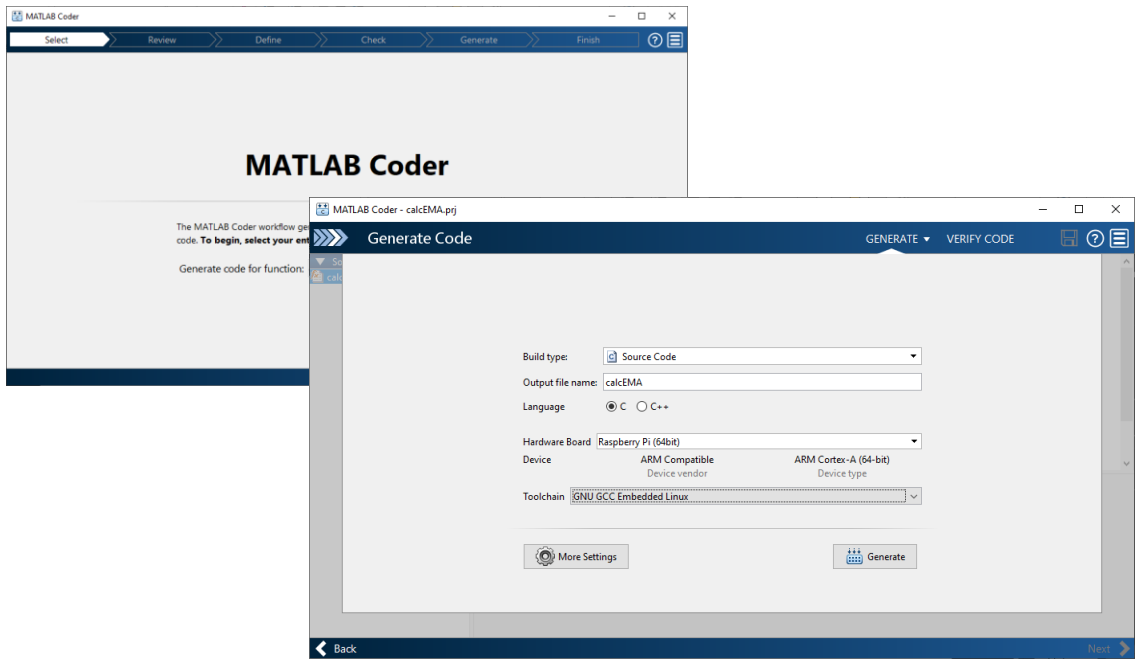
Practice #1: Exponential Moving Average

Procedure

1. Study `calcEMA` function to calculate Exponential Moving Average.
2. Write `test_calcEMA` script to invoke `calcEMA` function.

```
data = rand(1,100);
N = uint32(8);
ema = calcEMA(data, N)
```

3. Choose App tab, then start MATLAB Coder App to generate C code for Raspberry Pi (64bit) target:



4. Use `scp` command to copy generated C code (*.c and *.h) from `codegen/lib/calcEMA` to Raspberry Pi.
5. Prepare `Makefile` to build `ema_app`:

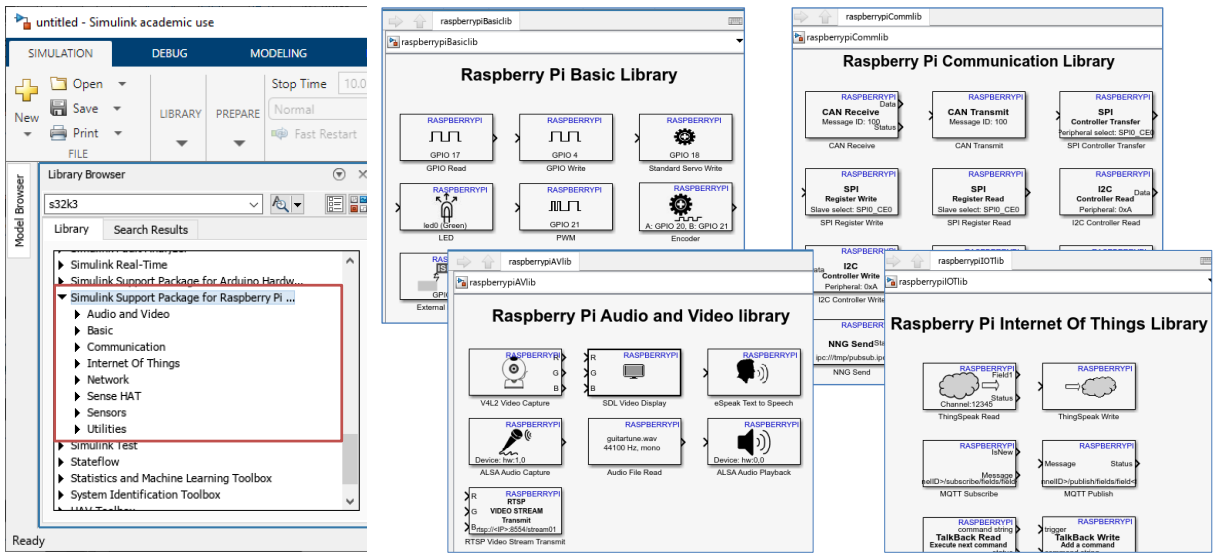
```
LIBS = -lm
TARGET = ema_app
SRCS = ema_app.c calcEMA_initialize.c calcEMA_terminate.c calcEMA.c
```

6. Build with `make`, then verify results with `./ema_app`.
7. Modify `calcEMA.m` function, then generate code again.

```
arguments
    data(1,100) double;
    N(1,1) double;
end
```

8. Copy source files, build, and compare results with (6).

RPi hardware support package



Raspberry Pi Support from MATLAB

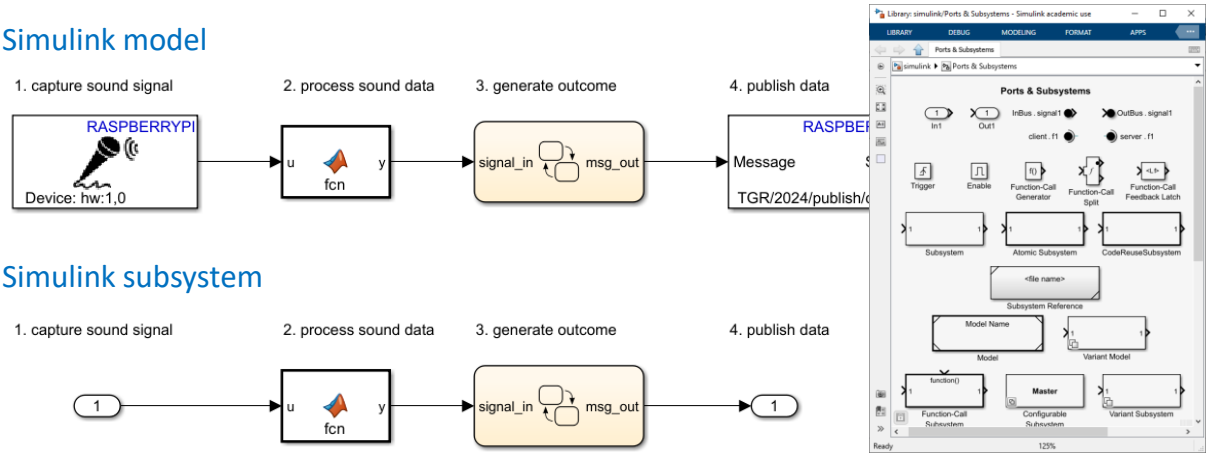
MATLAB Support Package for Raspberry Pi™ Hardware provides two ways of programming Raspberry Pi applications from MATLAB.

- **Interactive communication:** You can remotely communicate with a Raspberry Pi from a desktop installation of MATLAB or through a web browser with MATLAB Online. Acquire data from sensors and imaging devices connected to the Raspberry Pi and then analyze and visualize it in MATLAB.
- **Standalone execution:** With MATLAB Coder, you can develop standalone embedded applications for Raspberry Pi. Use the interactive communication to prototype and develop your MATLAB algorithm, then automatically generate equivalent C code and deploy it to the Raspberry Pi to run as a standalone application.

Simulink Support Package for Raspberry Pi™ lets you develop algorithms that run standalone on your Raspberry Pi. The support package extends Simulink with blocks to drive Raspberry Pi digital I/O and read and write data from them. After creating your Simulink model, you can simulate it and download the completed algorithm for standalone execution on the device. One particularly useful (and unique) capability offered by Simulink is the ability to tune parameters live from your Simulink model while the algorithm runs on the hardware.

- Library of Simulink blocks that connect to Raspberry Pi I/O, such as audio input and output, video input and display, GPIO read and write, and ThingSpeak read and write
- Hardware setup screens to configure Raspberry Pi hardware and Wi-Fi network interface
- Customization of existing Raspbian OS images to make it compatible with the Simulink Support Package
- Publish and subscribe blocks for MQTT client support for machine-to-machine and IoT applications
- Model deployment for standalone operation

Code generation with Simulink



Simulink modeling

A model is an abstract and simplified description of a system using mathematical equations and diagrams. A block diagram is a visual representation of a model in the Simulink Editor. The editor allows you to add blocks selected from block libraries representing elementary model components. A **block** is a basic modeling construct of the Simulink Editor. Add blocks from the built-in Simulink libraries to perform specific operations. You can also create custom blocks. Some blocks have input signals, output signals, and states. Most blocks have parameters that you use to specify block behavior. Blocks are connected to each other with signal and event lines to visually construct the model equations.

Virtual blocks organize and provide graphical hierarchy in a model. During model simulation, Simulink expands the blocks in place before execution, a process known as flattening. This expansion is similar to the way macros work in a programming language such as C or C++.

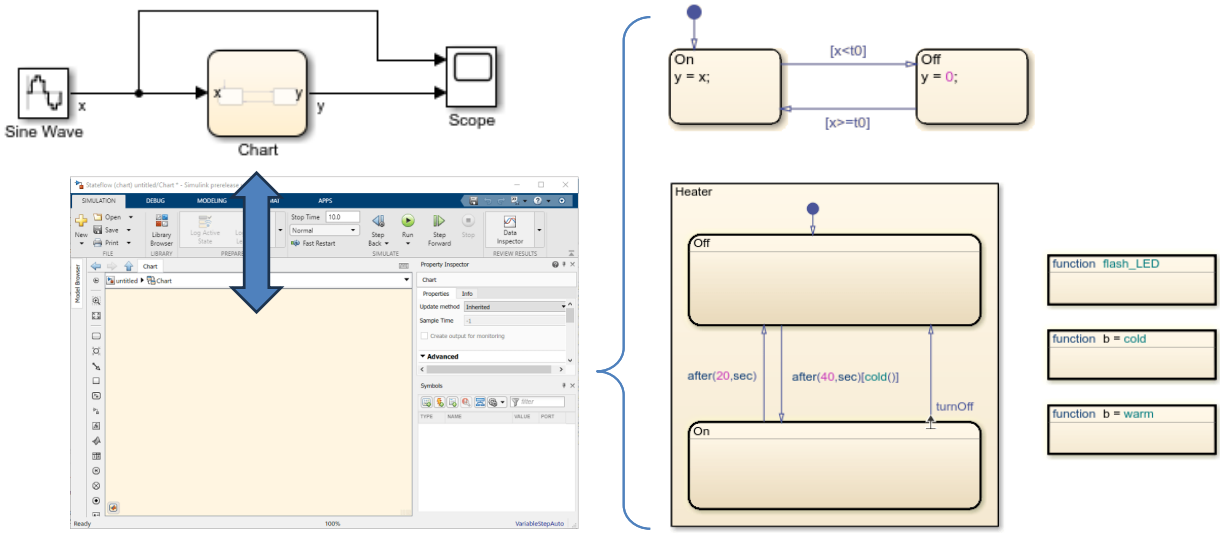
- **Virtual Subsystem** to encapsulate related and functional parts within a larger model.
- **Inport** and **Outport** to move data (signals) and events (function calls) from outside a Subsystem block or Model block to within the block, and vice versa.
- **Bus Creator**, **Bus Assignment**, **Bus Selector** to combine signals into a virtual bus and manage the routing of signals around a complex block diagram.

Nonvirtual blocks provide execution control and graphical hierarchy in a model.

- **Atomic Subsystem** holds blocks being executed as a single unit (atomic execution) at each time step.
- **Enabled** and/or **Triggered Subsystem** whose execution is controlled by external data from a signal
- **Function-Call Subsystem** whose execution is controlled by an event from an initiator.

https://www.mathworks.com/help/simulink/slref/simulink-concepts-models.html

Stateflow



Stateflow

Stateflow is a product that provides a graphical language that includes state transition diagrams, flow charts, state transition tables, and truth tables. You can use Stateflow to describe how MATLAB algorithms and Simulink models react to input signals, events, and time-based conditions. With Stateflow, you model combinatorial and sequential decision logic that can be simulated as a block within a Simulink model or executed as an object in MATLAB. Graphical animation enables you to analyze and debug your logic while it is executing. Edit-time and run-time checks ensure design consistency and completeness before implementation.

Procedure to model algorithm as finite state machines in Stateflow:

1. Construct and Run a Stateflow Chart
2. Define Chart Behavior by Using State Actions and Transition Labels
3. Create a Hierarchy to Manage System Complexity
4. Model Synchronous Subsystems by Using Parallel Decomposition
5. Synchronize Parallel States by Broadcasting Events
6. Monitor Chart Activity by Using Active State Data
7. Schedule Chart Actions by Using Temporal Logic

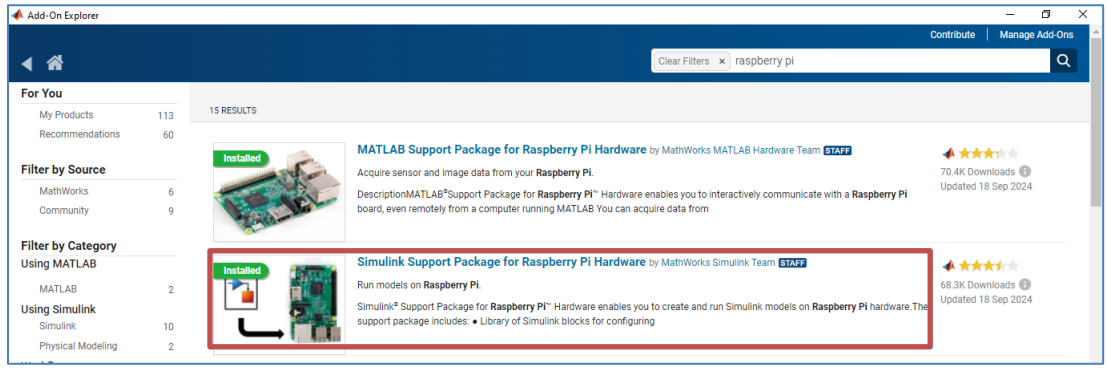
In Simulink, Chart blocks (from Stateflow) execute based on a discrete-time sampling period. This means they are triggered to update their state and execute actions at specific intervals. The execution schedule is influenced by the following factors: **Simulink Solver** (global sample time), Chart Properties (block sample time), and State Transition Conditions (conditions that triggering actions).

https://www.mathworks.com/help/stateflow/getting-started.html

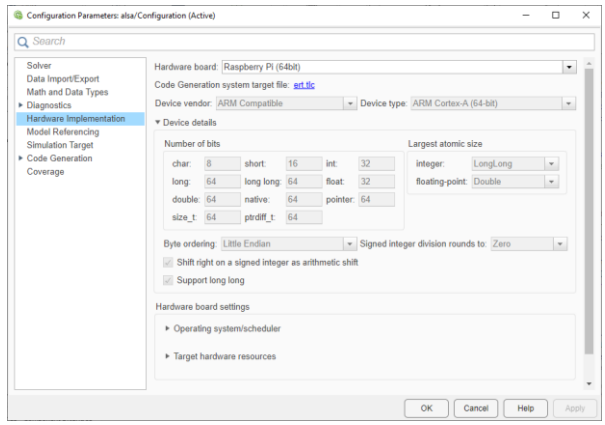
Practice #2: code generation with Simulink

Procedure

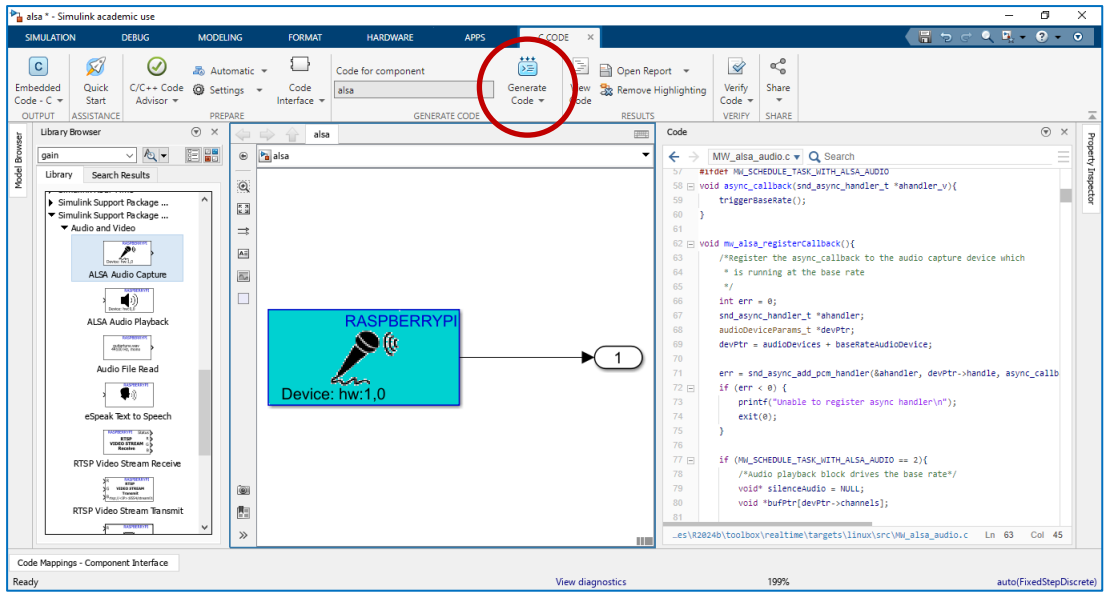
1. Choose Add-ons > Get Hardware Support Packages, and install Simulink Support Package for Raspberry Pi:



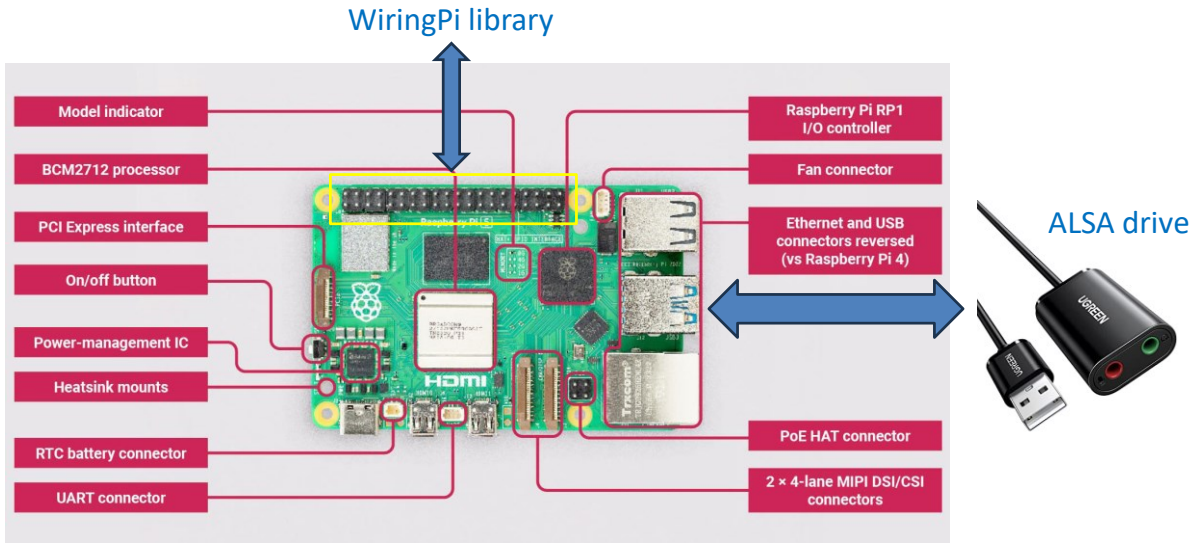
2. Create a Simulink blank model.
3. Choose Library Browser, then add ALSA Audio Capture and Outport blocks.
4. Choose Modeling > Model Settings, then choose Hardware Implementation:



5. From Apps > Embedded Coder, change from Build to Generate Code:



RPi hardware programming



Using Raspberry Pi hardware

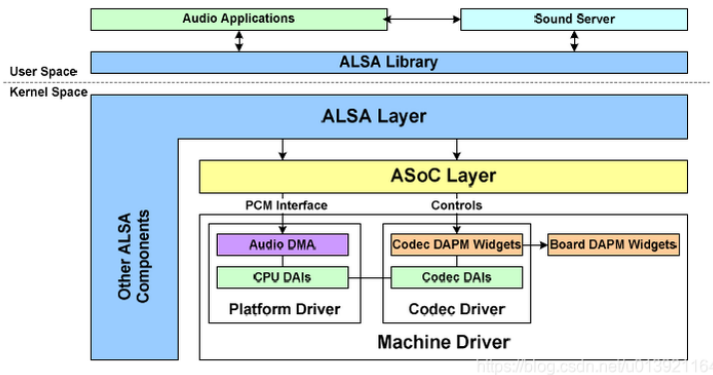
Hardware devices are accessed by the user through special device files. These files are grouped into the **/dev** directory, and system calls open, read, write, close, lseek, mmap etc. are redirected by the operating system to the device driver associated with the physical device. The Raspberry Pi supports a variety of device driver models, each designed for specific types of devices and functionalities. Here are the major device driver models:

- **Block Devices:** devices that store data in blocks, such as hard drives. Operations are performed on entire blocks, rather than individual bytes.
- **Character Devices:** devices that handle data streams, such as serial ports, keyboards, and mice. Data is transferred in a stream of bytes, without any fixed block structure.
- **Network Devices:** network interfaces, such as Wi-Fi. Handle network packets and provide communication with other devices on the network.
- **Input Devices:** input devices, such as keyboards, mice, and touchscreens. Handle input events and provide information to the operating system.
- **Output Devices:** output devices, such as displays, speakers, and printers. Handle output data and produce the desired output.
- **Virtual Devices:** virtual devices that exist in software, rather than in hardware. Can be used to emulate hardware devices or provide additional functionality.

The Raspberry Pi also supports other device types, such as audio devices, video devices, and sensor devices.Each device type may have its own specific driver model.

GPIO (General-Purpose Input/Output) pins are versatile pins on the Raspberry Pi that can be configured as inputs or outputs to interact with external devices. The Raspberry Pi's OS typically include built-in GPIO libraries that provide a convenient interface for accessing and controlling the GPIO pins. For example, WiringPi is a popular C library that provides a simple and intuitive interface for controlling GPIO pins.

Alsa driver in Raspberry Pi




```
#include <alsa/asoundlib.h>
```

1. Open audio device
   `snd_pcm_open()`
2. Configure audio parameters
   `snd_pcm_hw_params()`
3. Allocate memory buffer
4. Prepare audio device
   `snd_pcm_prepare()`
5. Start capture
   `snd_pcm_start()`
6. Read audio data
   `snd_pcm_readi()`
7. Stop capture
   `snd_pcm_drain()`
   `snd_pcm_stop()`

ALSA driver

The audio device driver model in the Raspberry Pi is a complex system that handles the interaction between OS and the underlying audio hardware. It's responsible for tasks like capturing audio from microphones and playing audio through speakers.

- ALSA (Advanced Linux Sound Architecture): abstraction layer for audio devices in Linux.
- Device Tree Overlay: A mechanism to add or modify device tree information at runtime.
- Audio Codec Driver: Handles direct communication with the audio codec hardware.
- Platform-Specific Audio Driver: Responsible for interfacing with the Raspberry Pi's specific audio hardware (e.g., GPIO pins, I2C bus).

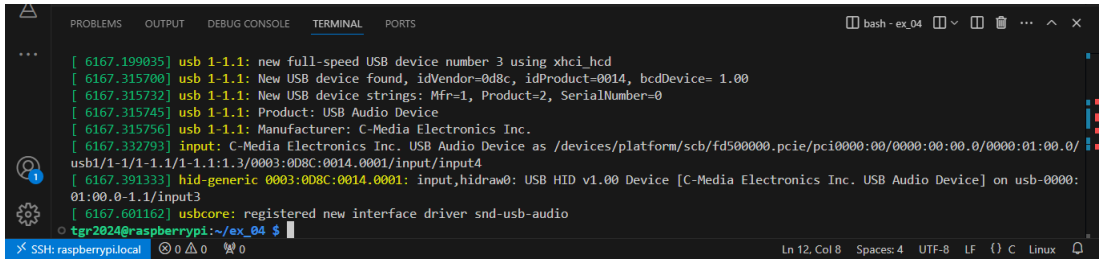
Additional sound capabilities can be added to a Raspberry Pi using a USB audio device. This can improve the sound quality as well as adding a microphone input. There are a number of different audio devices you can use with the Raspberry Pi's USB ports. Ugreen USB Audio Adapter includes a stereo headphone output jack and a mono microphone input jack. It can bypass the defective sound card or the faulty 3.5mm audio port of your laptop/desktop system and regain the audio function for you.

https://www.raspberrypi.com/documentation/accessories/audio.html
https://kernel.org/doc/html/latest/sound/index.html
https://www.alsa-project.org/alsa-doc/alsa-lib/index.html
https://soundprogramming.net/programming/alsa-tutorial-1-initialization/

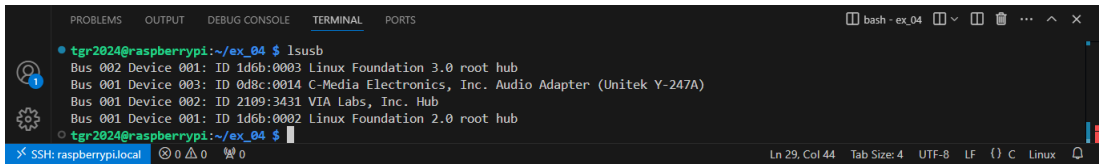
Practice #3: sound detection

Procedure

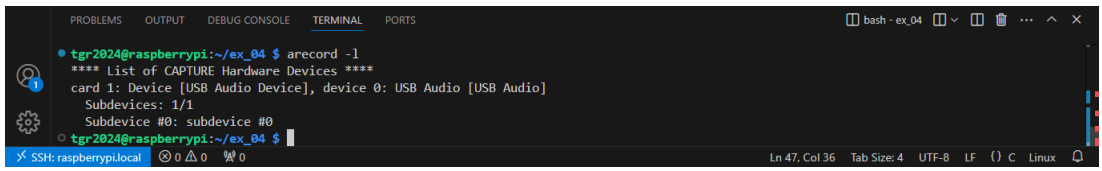
1. Install required libraries: `sudo apt install libasound2 libasound2-dev`
2. Plug USB sound adapter to Raspberry Pi's USB port.
3. Check status of driver: `sudo dmesg`



4. Check status of USB device: `lsusb`



5. Check device ID detected by ALSA driver: `arecord -l`



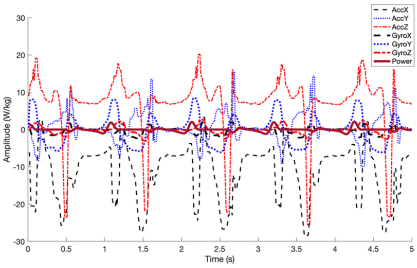
6. Prepare Makefile for ex_13/sound_app.c:

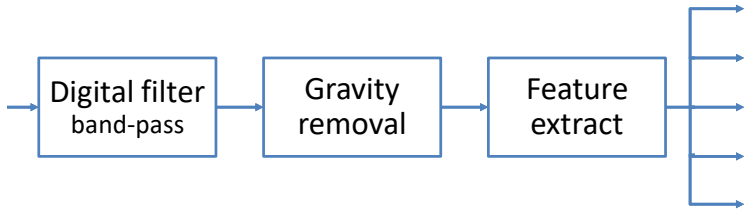
```
LIBS = -lm -lasound
TARGET = sound_app
SRCS = sound_app.c
```

7. Build with `make`, then execute with `./sound_app plughw:1,0 48000`
8. Plug USB audio with AUX cable with computer, then play sound.

Signal processing

IMU signals





- High-frequency noise
- Low-frequency drift
- Effect of gravity

- Windowing
- Adaptive
- Frequency-domain
- Joint time-frequency domain
- Fusion

Signal processing concepts

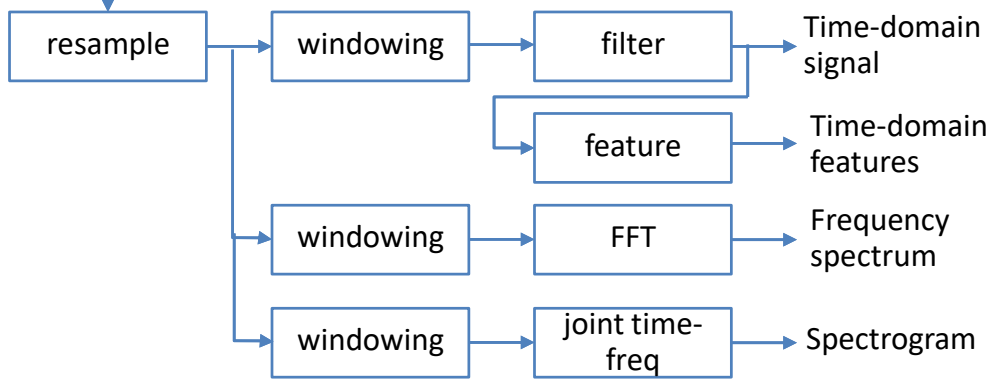
Signal processing is an electrical engineering subfield that focuses on analyzing, modifying and synthesizing signals, such as sound, images, potential fields, seismic signals, altimetry processing, and scientific measurements. Digital signal processing is the processing of digitized discrete-time sampled signals. Typical arithmetical operations include **fixed-point** and **floating-point**, real-valued and complex-valued, multiplication and addition. Sample-based processing enables low-latency processes and applications that require scalar processing. Frame-based processing enables higher throughput in exchange for latency.

Simulink is a powerful tool for developing signal processing applications for Raspberry Pi. It provides a graphical environment where you can design, simulate, and test your algorithms using a drag-and-drop interface. This includes tasks such as filtering, noise reduction, feature extraction, and spectral analysis. By leveraging Simulink's built-in blocks and libraries, you can efficiently develop and deploy signal processing systems on Raspberry Pi hardware.

- **Signal Processing Toolbox** provides functions and apps to manage, analyze, preprocess, and extract features from uniformly and nonuniformly sampled signals. The toolbox includes tools for filter design and analysis, resampling, smoothing, detrending, and power spectrum estimation.
- **DSP System Toolbox** offers a library of signal processing algorithm blocks in Simulink for filters, transforms, and linear algebra. These blocks process streaming input signals as individual samples or as collections of samples called frames.

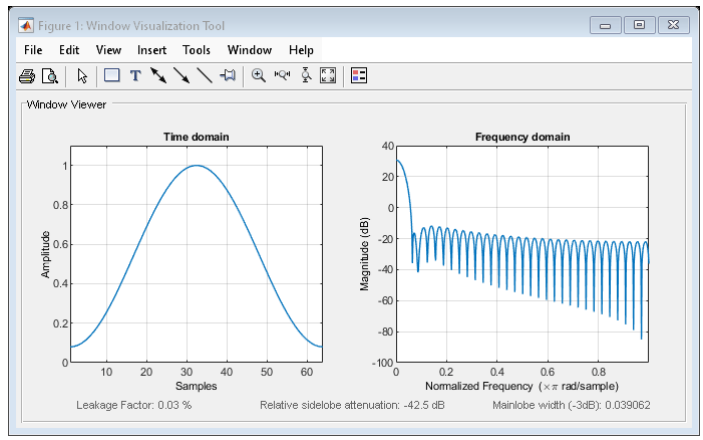
https://www.mathworks.com/products/signal.html
https://www.mathworks.com/products/dsp-system.html

Signal processing → ML features



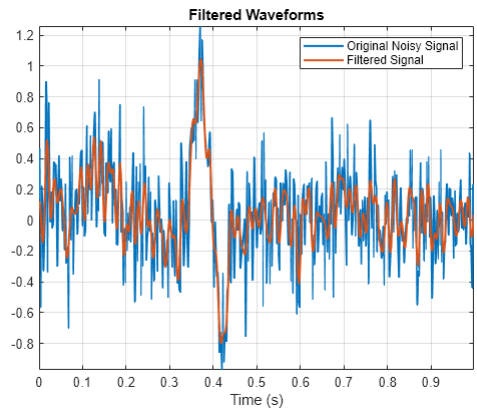
Windows

In both digital filter design and spectral estimation, the choice of a windowing function can play an important role in determining the quality of overall results. The main role of the window is to damp out the effects of the Gibbs phenomenon that results from truncation of an infinite series.

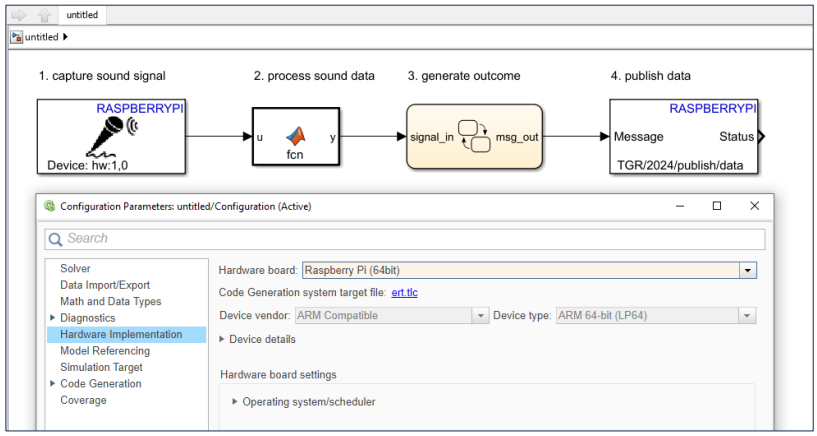


Filter

Filters can be used to shape the signal spectrum in a desired way or to perform mathematical operations such as differentiation and integration. In what follows you will learn some practical concepts that will ease the use of filters when you need them.



Signal processing flow

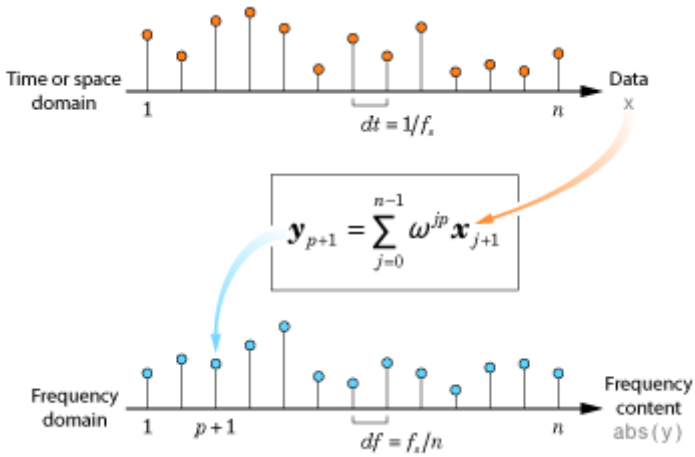


Key parameters

- Hardware target
  Raspberry Pi
- Sample time
  → capture window
- Algorithm
  → Simulink Math
  → DSP System Toolbox
  → MATLAB Function
- Data types
  → Data Type Conversion
- Data logics
  → JSON commands
  → Stateflow

Fast Fourier Transform

Transforms and filters are tools for processing and analyzing discrete data, and are commonly used in signal processing applications and computational mathematics. When data is represented as a function of time or space, the Fourier transform decomposes the data into frequency components. The `fft` function uses a fast Fourier transform algorithm that reduces its computational cost compared to other direct implementations.



`Y = fft(X)` computes the discrete Fourier transform (DFT) of X using a fast Fourier transform (FFT) algorithm. Y is the same size as X.

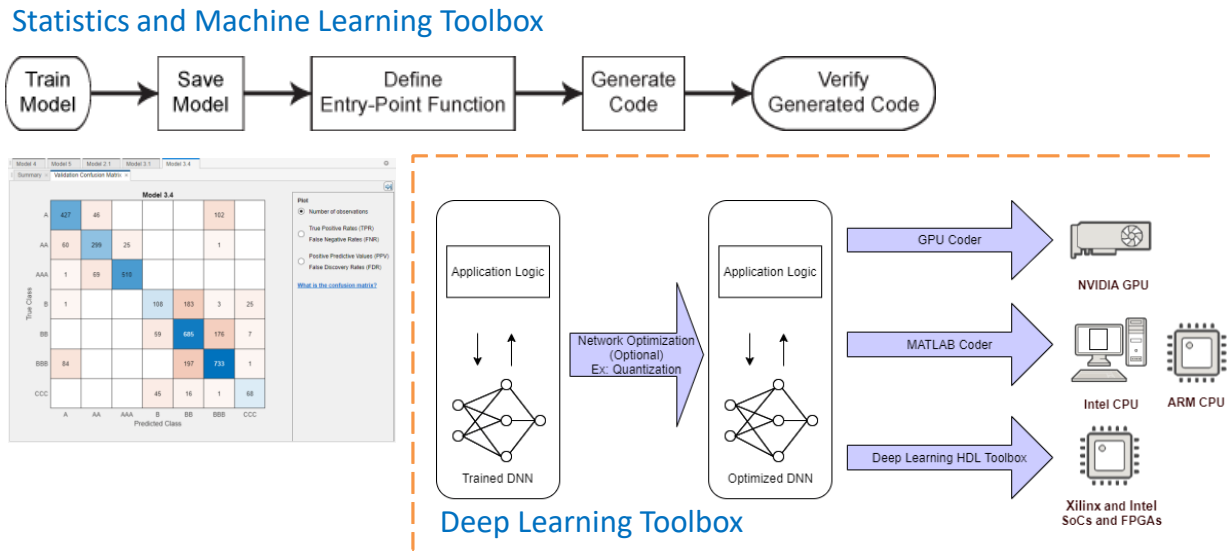
https://www.mathworks.com/help/releases/R2024b/matlab/math/basic-spectral-analysis.html
https://www.mathworks.com/help/releases/R2024b/matlab/ref/fft.html

Practice #3: real-time sound processing

Procedure

1. Use MATLAB Coder workflow to generate C source files for **sound_freq.m**
2. Use `scp` command to copy *.c and *.h from **codegen/lib/sound_freq** to Raspberry Pi
3. Modify code to include **sound_freq()** function
4. Prepare **Makefile** for build **sound_app**
5. Build and test with `./sound_app`

Machine learning with MATLAB Coder



Machine learning in MATLAB

Statistics and Machine Learning Toolbox provides functions and apps to describe, analyze, and model data. You can use descriptive statistics, visualizations, and clustering for exploratory data analysis; fit probability distributions to data; generate random numbers for Monte Carlo simulations, and perform hypothesis tests. Regression and classification algorithms let you draw inferences from data and build predictive models either interactively, using the Classification and Regression Learner apps, or programmatically, using AutoML. For multidimensional data analysis and feature extraction, the toolbox provides principal component analysis (PCA), regularization, dimensionality reduction, and feature selection methods that let you identify variables with the best predictive power.

Deep Learning Toolbox provides functions, apps, and Simulink blocks for designing, implementing, and simulating deep neural networks. The toolbox provides a framework to create and use many types of networks, such as convolutional neural networks (CNNs) and transformers. You can visualize and interpret network predictions, verify network properties, and compress networks with quantization, projection, or pruning. The toolbox lets you interoperate with other deep learning frameworks. You can import PyTorch®, TensorFlow™, and ONNX™ models for inference, transfer learning, simulation, and deployment. You can also export models to TensorFlow and ONNX.

Both Statistics and Machine Learning Toolbox and Deep Learning Toolbox support code generation C/C++, CUDA, and HDL code on target hardware.

https://www.mathworks.com/help/stats/code-generation.html
https://www.mathworks.com/help/deeplearning/code-generation.html