MATLAB Coder workshop

code generation for Raspberry Pi

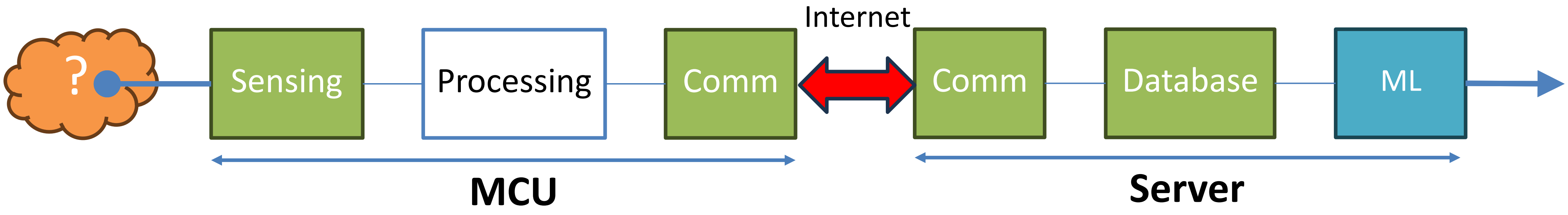
ผศ.ดร.ศุภชัย วรพจน์พิศุทธิ์

ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์

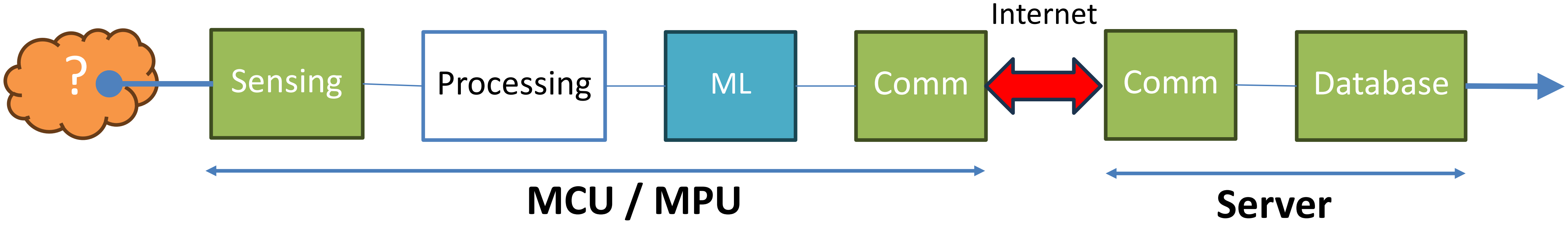
ม.ธรรมศาสตร์

AIoT architecture

Cloud-computing approach

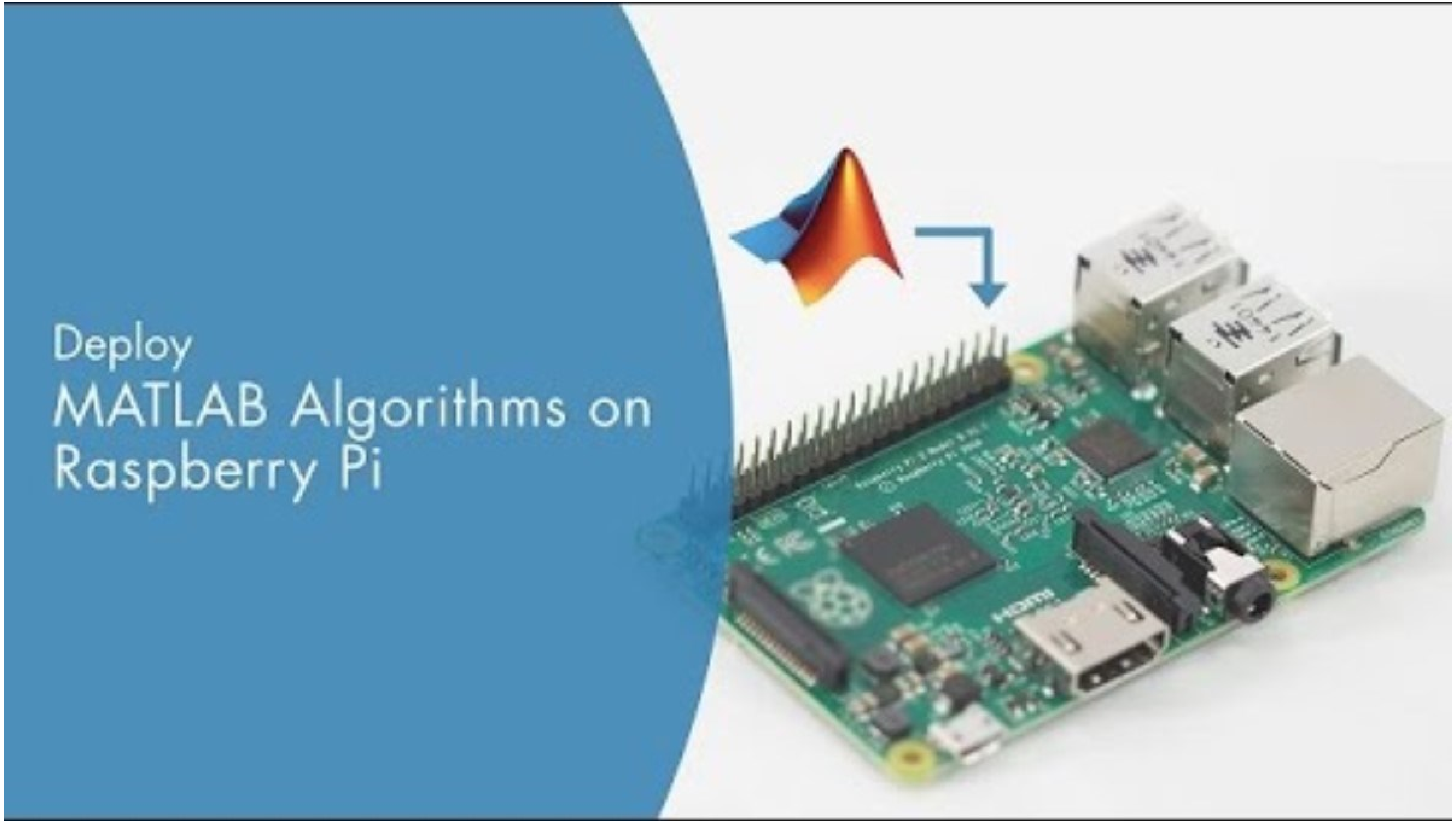


Edge-computing approach



Audio signal processing with RPi + MATLAB

Algorithm development with RPi + MATLAB



Deploy
MATLAB Algorithms on
Raspberry Pi

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

- Model

Simulink + custom block

- Model
- Subsystem
- Library

MATLAB coder + dev toolchain

- Application
- Function
- Library

Embedded coder

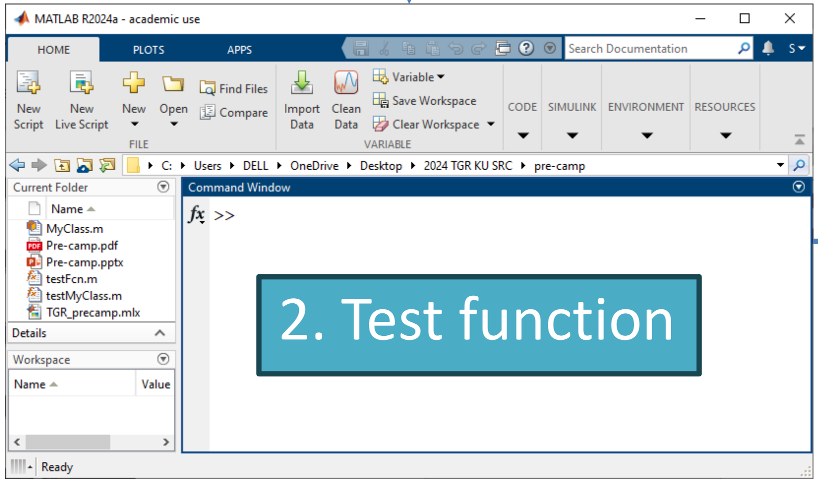
- Project
- Function

MATLAB Coder workflow

testFcn.m file

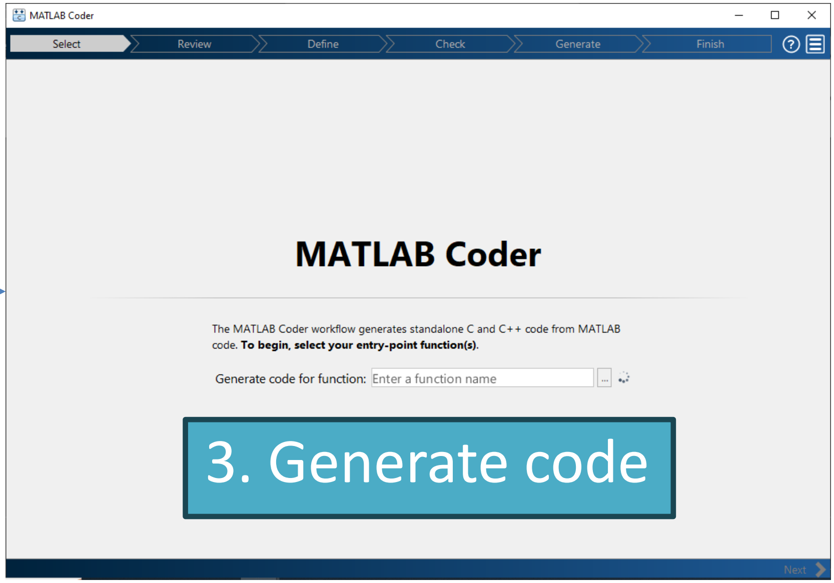
```
function outArg = testFcn(inArg)
%#codegen
outArg = mean(inArg);
```

1. Write function



2. Test function

MATLAB Coder App



MATLAB Coder

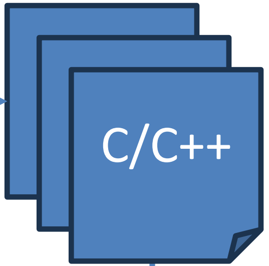
The MATLAB Coder workflow generates standalone C and C++ code from MATLAB code. **To begin, select your entry-point function(s).**

Generate code for function: Enter a function name

3. Generate code

1. Choose function
2. Define input type and dimension
3. Evaluate function → output
4. Check runtime issues
5. Generate C or C++ code

Source files



4. Write main

5. Build app

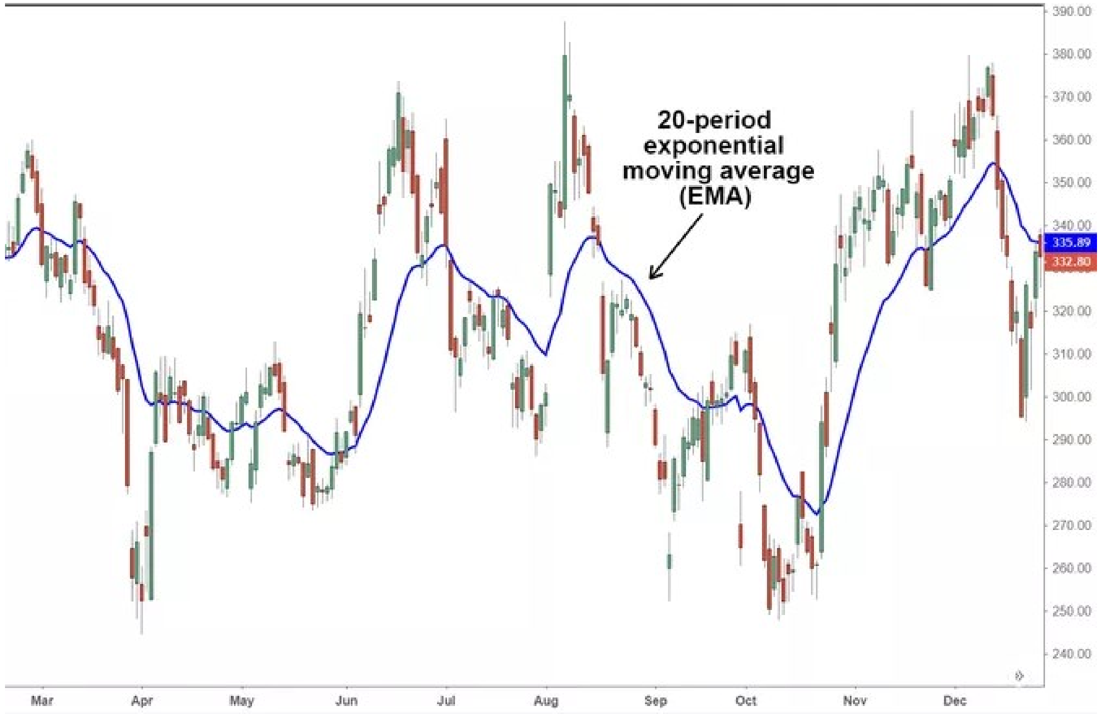
Practice #1: Exponential Moving Average

Formula for Exponential Moving Average (EMA)

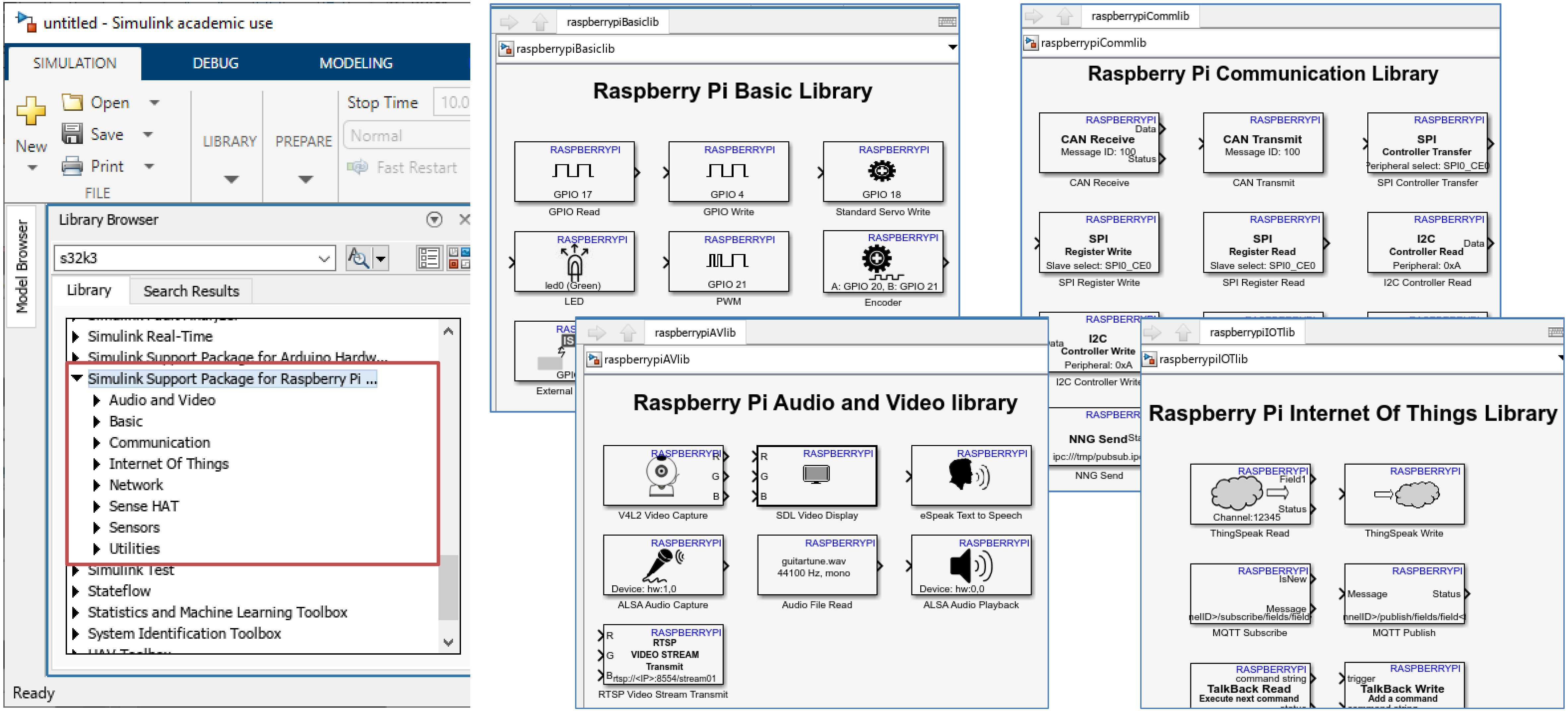
$$EMA_{\text{Today}} = \left(\text{Value}_{\text{Today}} * \left(\frac{\text{Smoothing}}{1 + \text{Days}}\right)\right) + EMA_{\text{Yesterday}} * \left(1 - \left(\frac{\text{Smoothing}}{1 + \text{Days}}\right)\right)$$

where:

$EMA = \text{Exponential moving average}$

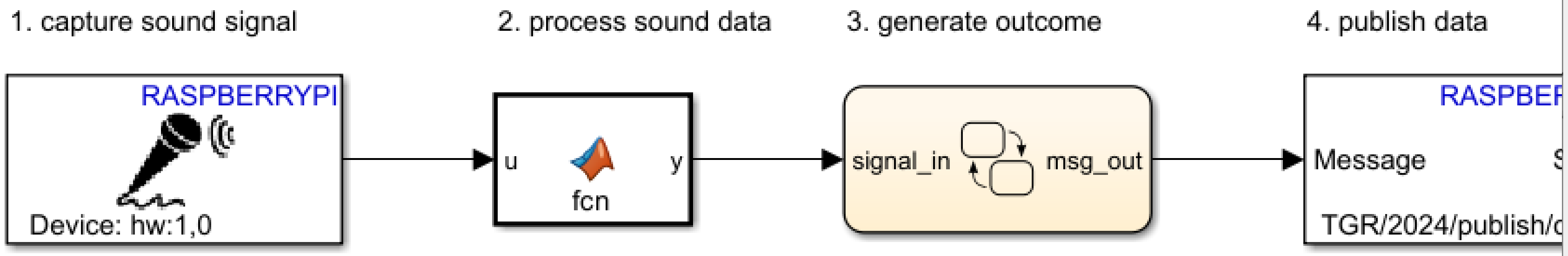


RPi hardware support package

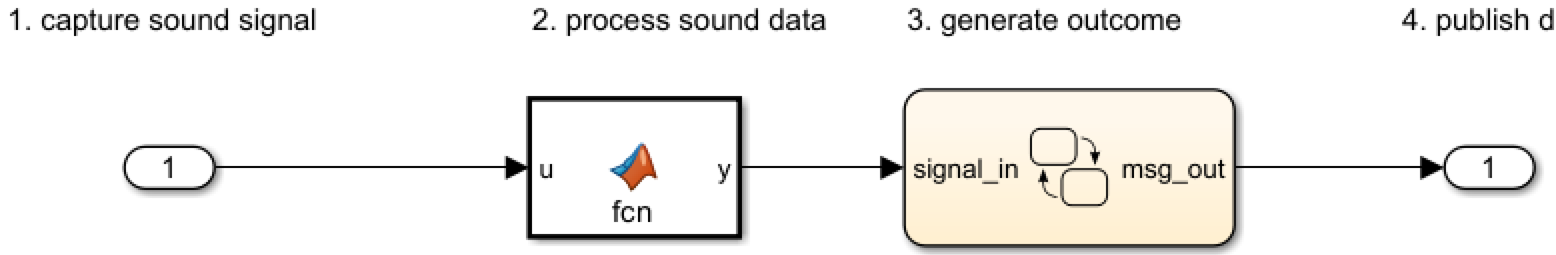


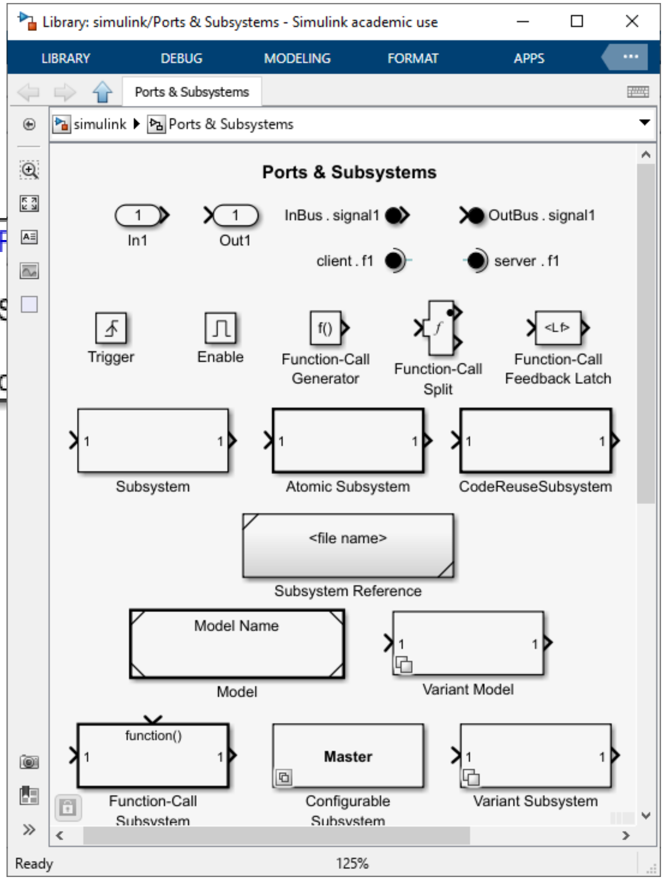
Code generation with Simulink

Simulink model

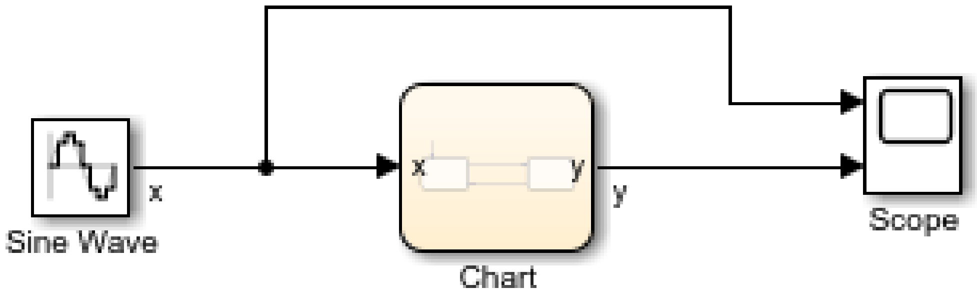


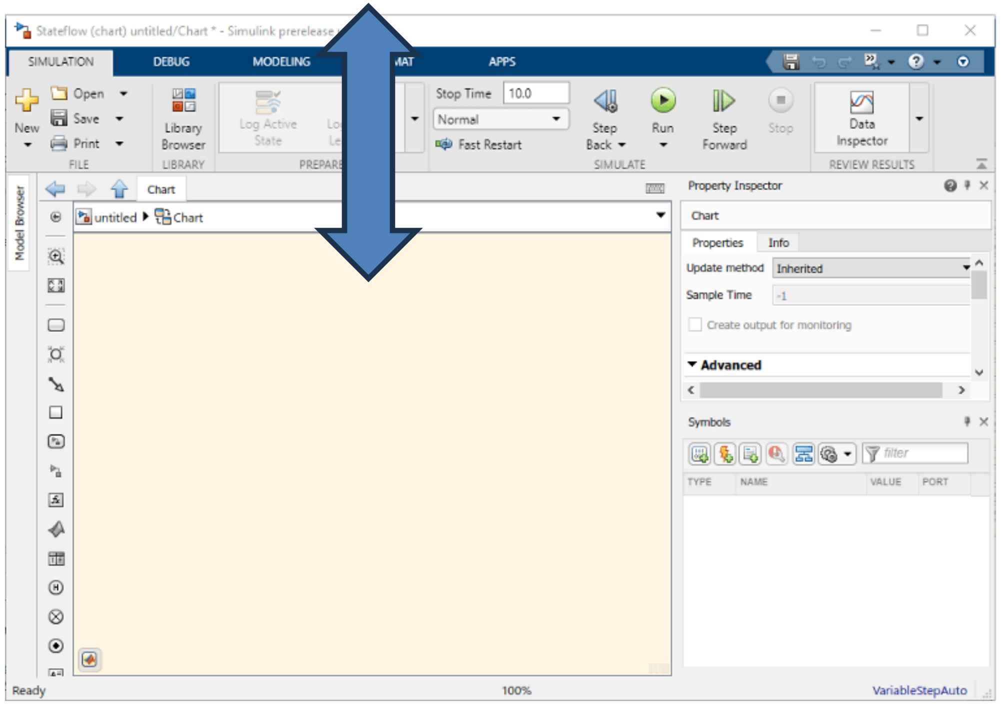
Simulink subsystem





Stateflow





Stop Time 10.0

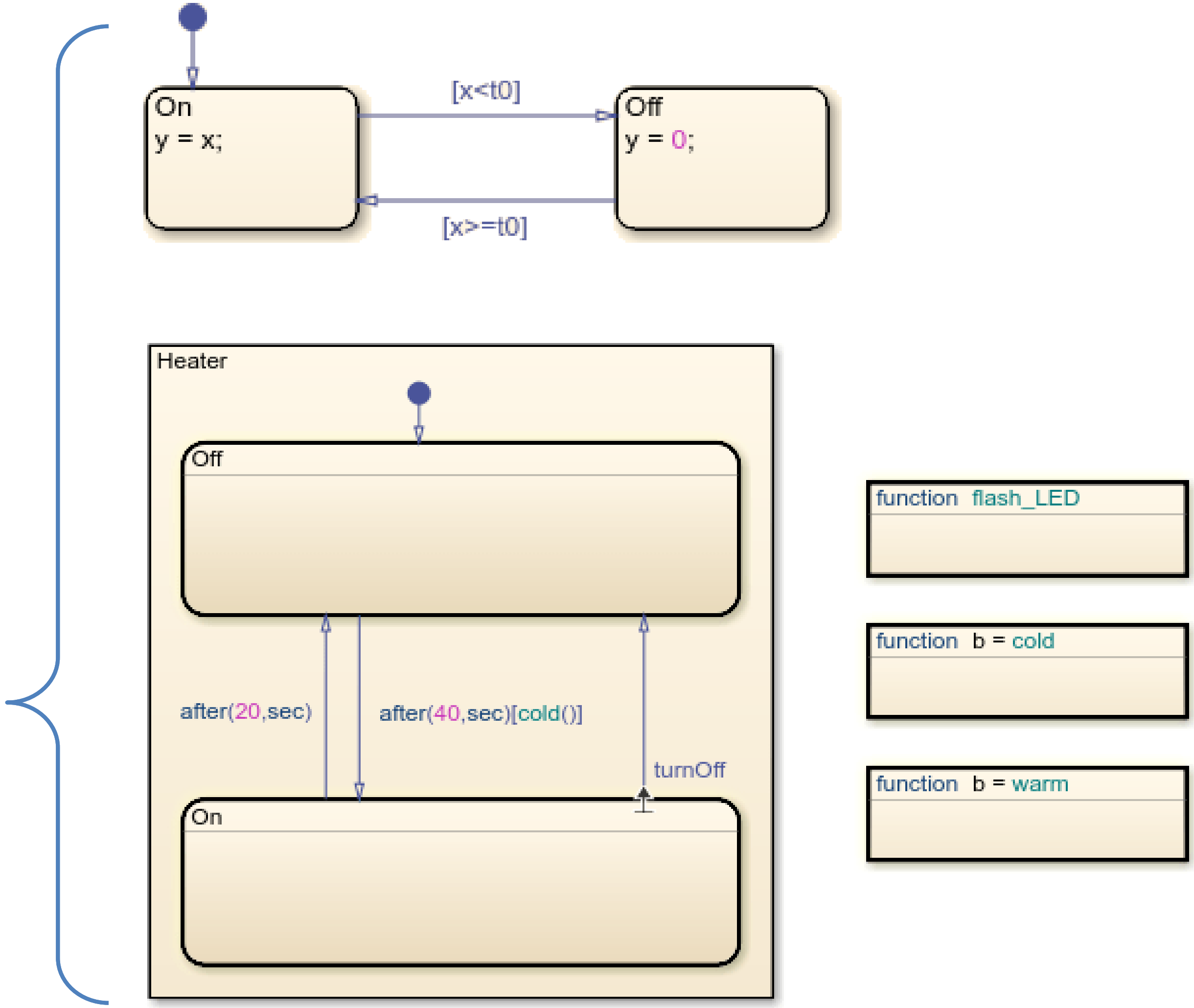
Normal

Chart

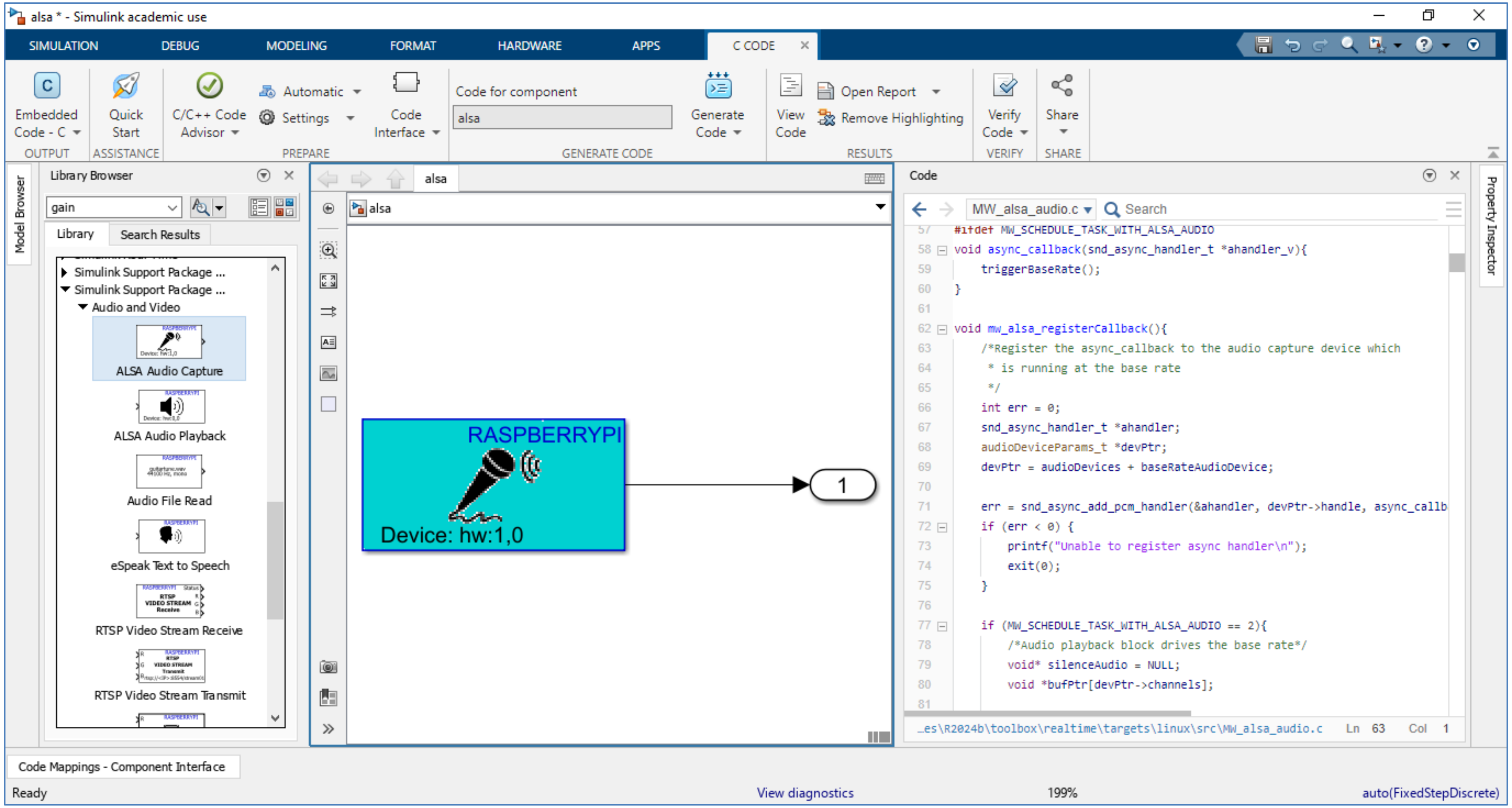
Update method Inherited

Sample Time -1

□ Create output for monitoring



Practice #2: code generation with Simulink

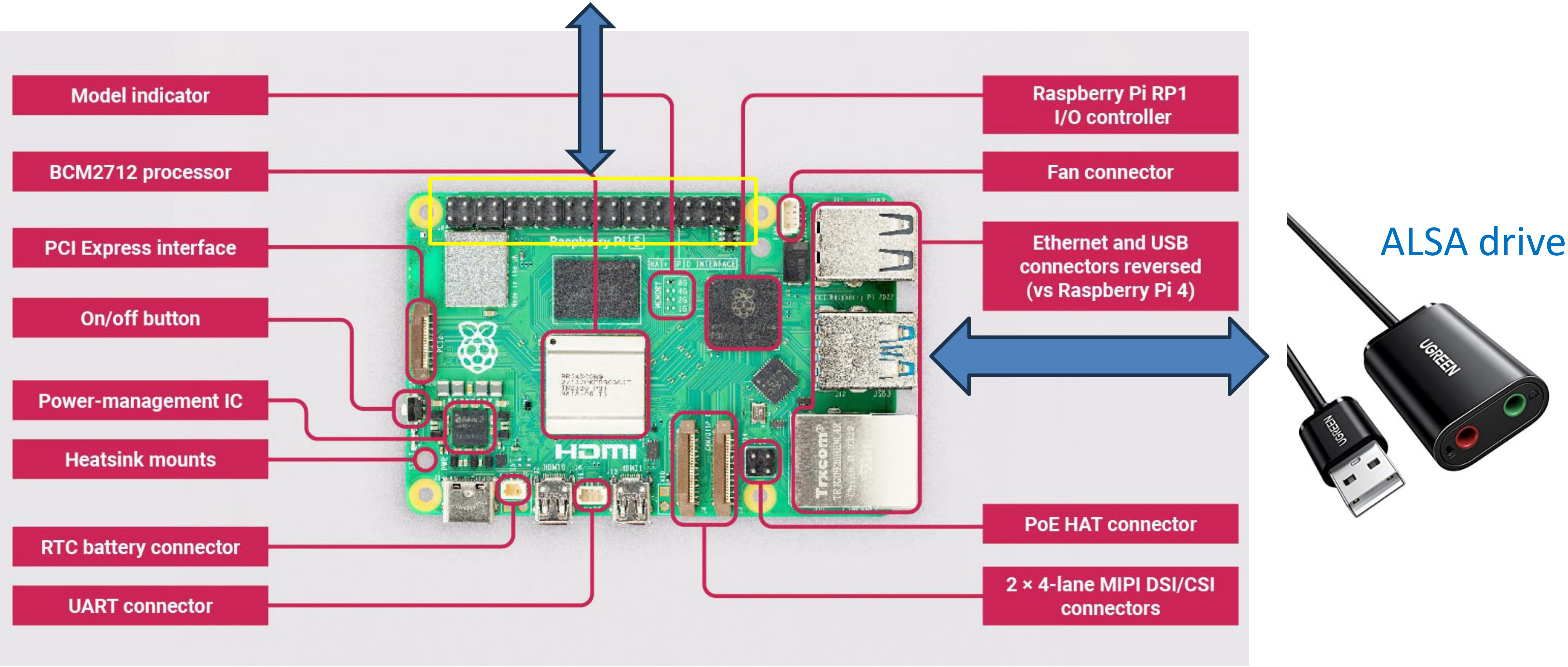


```c
#ifdef MW_SCHEDULE_TASK_WITH_ALSA_AUDIO
void async_callback(snd_async_handler_t *ahandler_v){
    triggerBaseRate();
}

void mw_alsa_registerCallback(){
    /*Register the async_callback to the audio capture device which
     * is running at the base rate
     */
    int err = 0;
    snd_async_handler_t *ahandler;
    audioDeviceParams_t *devPtr;
    devPtr = audioDevices + baseRateAudioDevice;

    err = snd_async_add_pcm_handler(&ahandler, devPtr->handle, async_callb
    if (err < 0) {
        printf("Unable to register async handler\n");
        exit(0);
    }

    if (MW_SCHEDULE_TASK_WITH_ALSA_AUDIO == 2){
        /*Audio playback block drives the base rate*/
        void* silenceAudio = NULL;
        void *bufPtr[devPtr->channels];

```

…es\R2024b\toolbox\realtime\targets\linux\src\MW_alsa_audio.c   Ln 63   Col 1

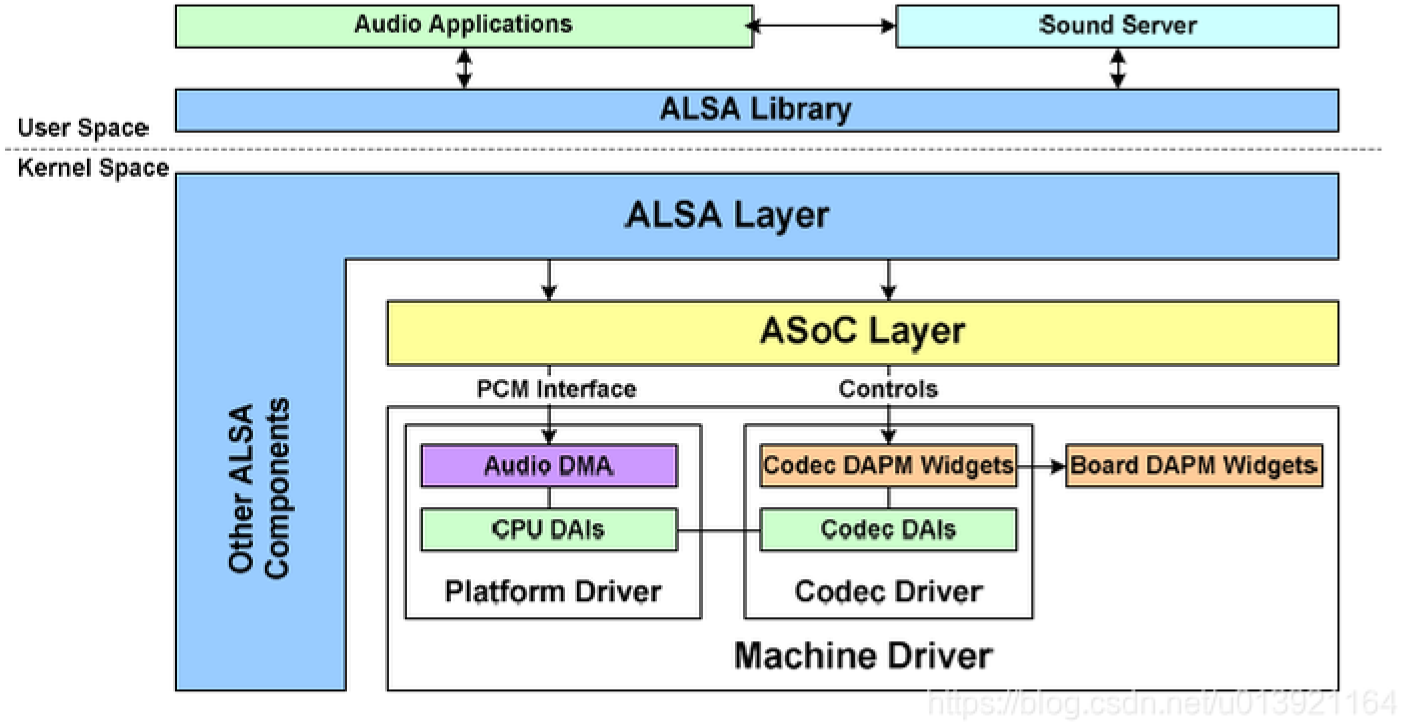
RPi hardware programming

WiringPi library



ALSA driver

Alsa driver in Raspberry Pi



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

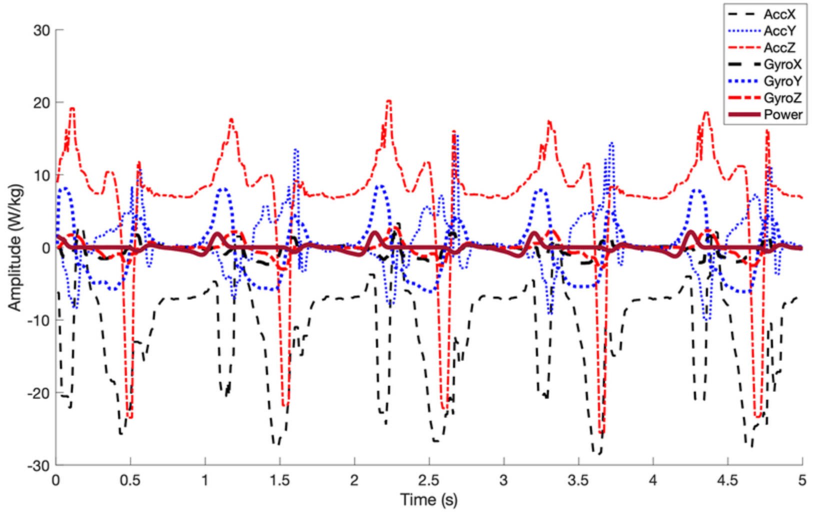
1. Open audio device
   `snd_pcm_open()`
2. Configure audio parameters
   `snd_pcm_hw_params()`
3. Allocate memory buffer
4. Prepare audio device
   `snd_pcm_prepare()`
5. Start capture
   `snd_pcm_start()`
6. Read audio data
   `snd_pcm_readi()`
7. Stop capture
   `snd_pcm_drain()`
   `snd_pcm_stop()`

Practice #3: sound detection

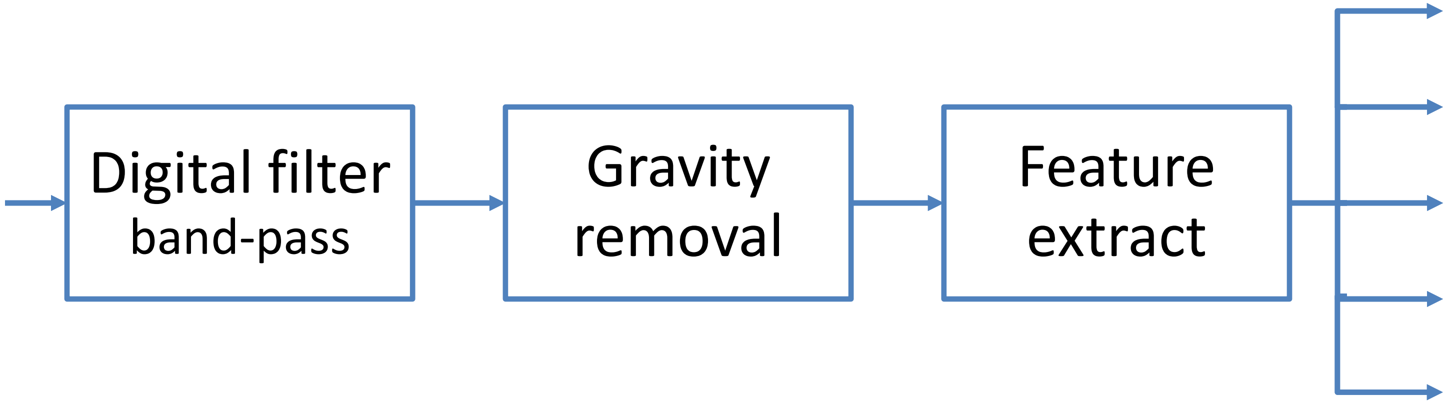
```
#include <alsa/asoundlib.h>

int main() {
   snd_pcm_open("plughw:1,0");

   snd_pcm_hw_params_alloca();
   ...
   snd_pcm_prepare();
   ...
   snd_pcm_readi();
   snd_pcm_readn();

}
```

Signal processing

IMU signals



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



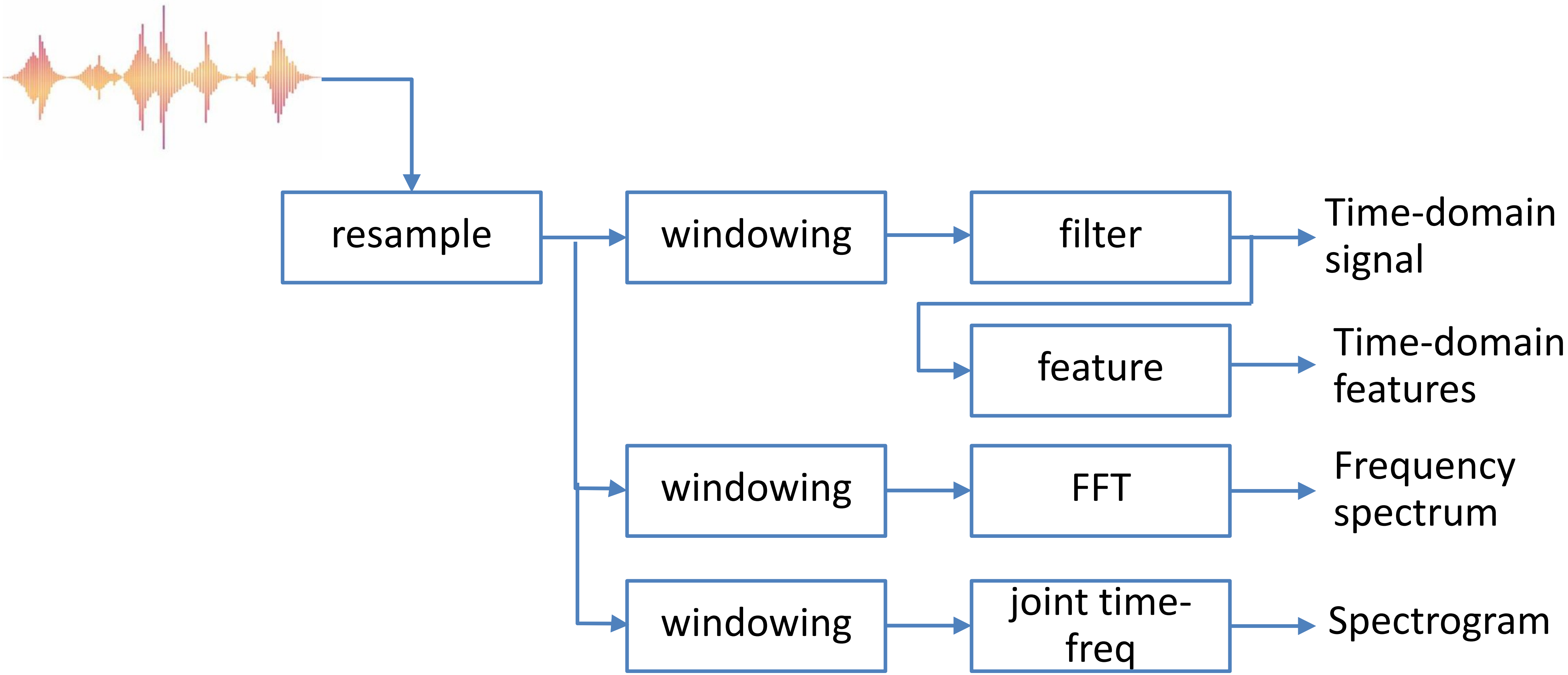
Digital filter band-pass

Gravity removal

Feature extract

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

Signal processing → ML features



resample

windowing

filter

Time-domain signal

feature

Time-domain features

windowing

FFT

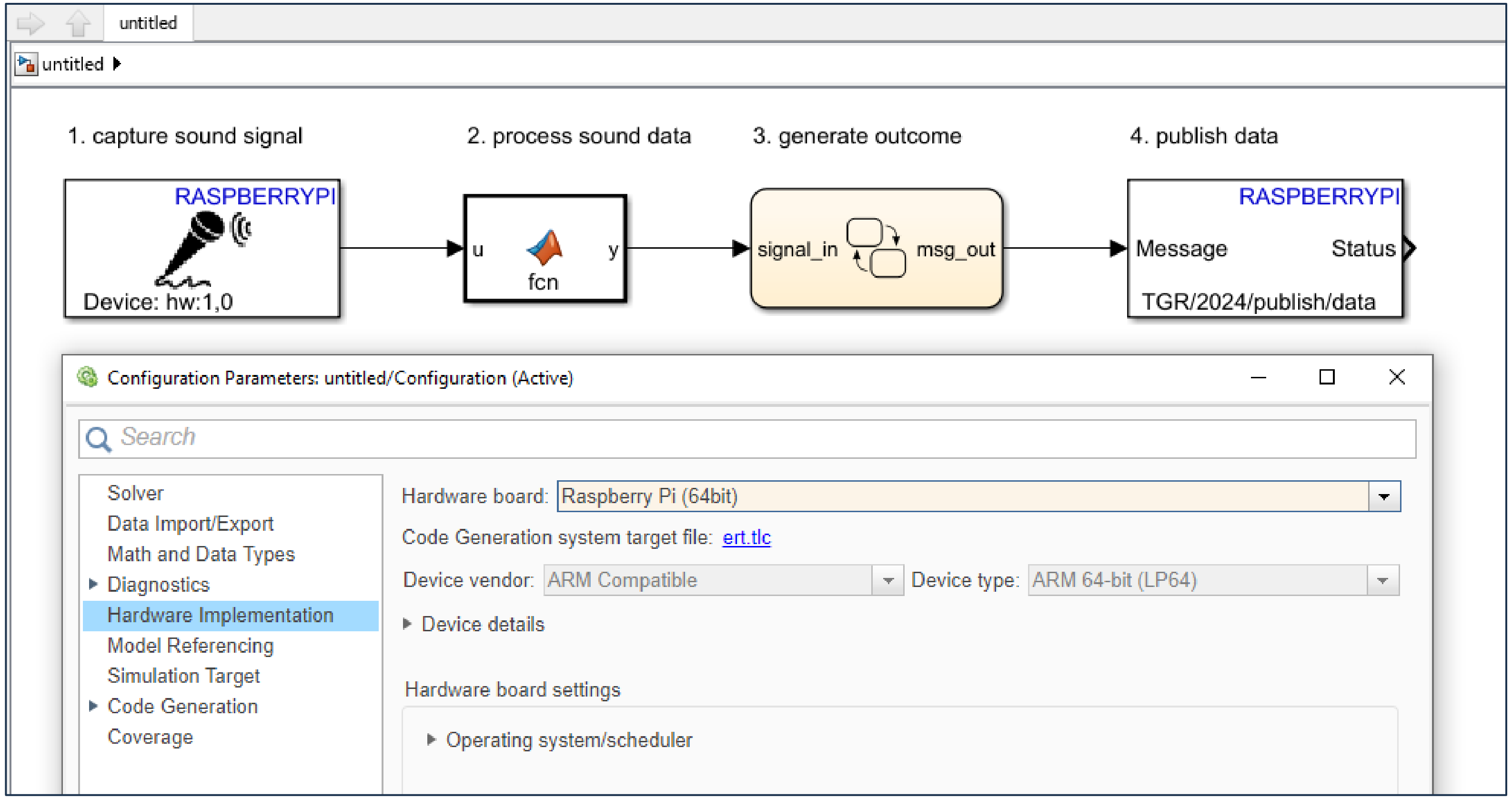
Frequency spectrum

windowing

joint time-freq

Spectrogram

Signal processing flow



Hardware board: Raspberry Pi (64bit)

Code Generation system target file: ert.tlc

Device vendor: ARM Compatible Device type: ARM 64-bit (LP64)

▸ Device details

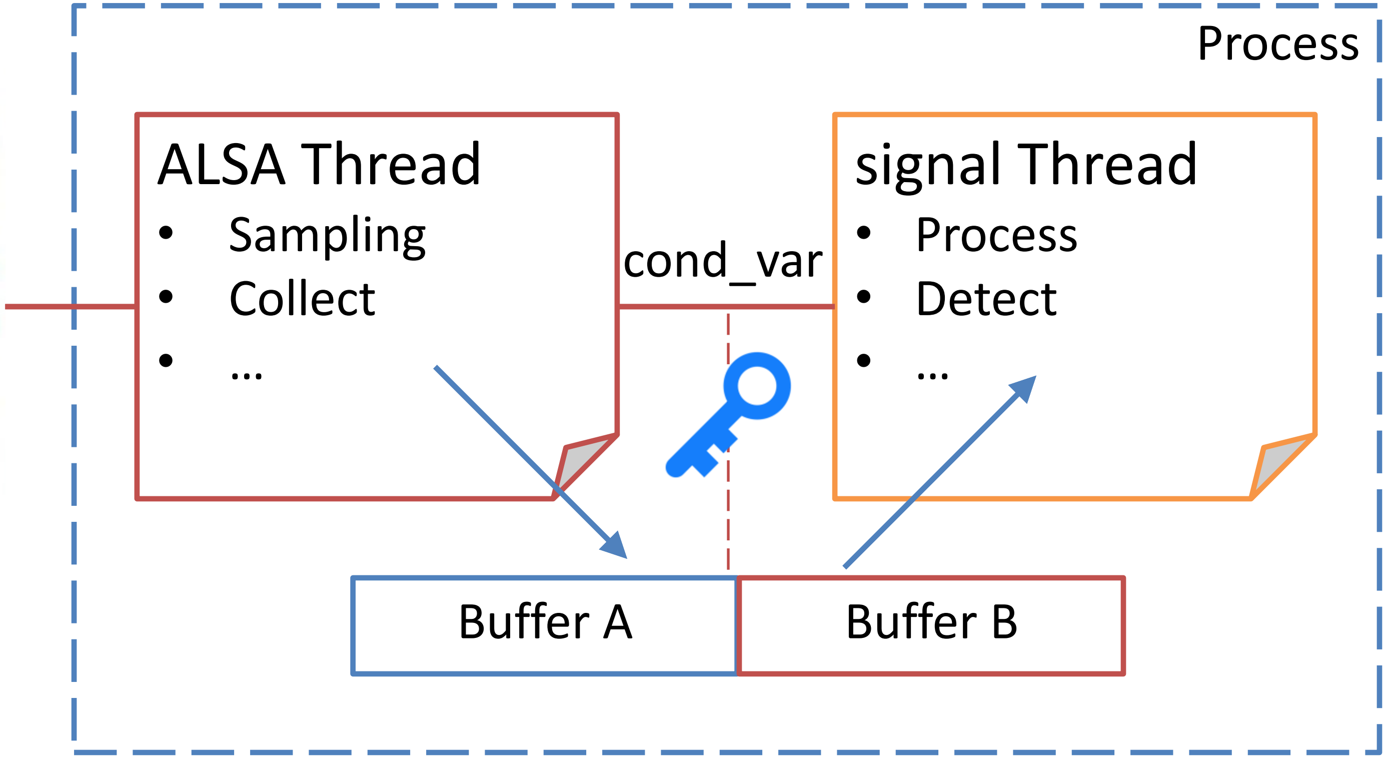
Hardware board settings

▸ Operating system/scheduler

Key parameters

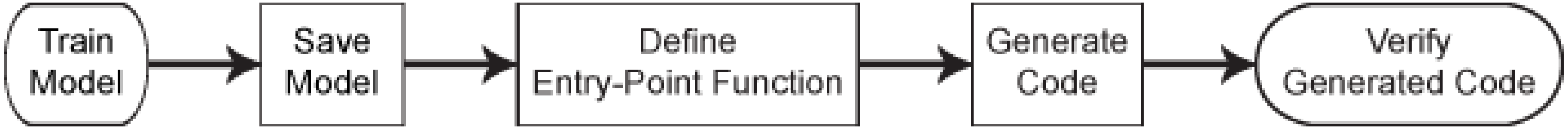
- Hardware target
  Raspberry Pi
- Sample time
  → capture window
- Algorithm
  → Simulink Math
  → DSP System Toolbox
  → MATLAB Function
- Data types
  → Data Type Conversion
- Data logics
  → JSON commands
  → Stateflow

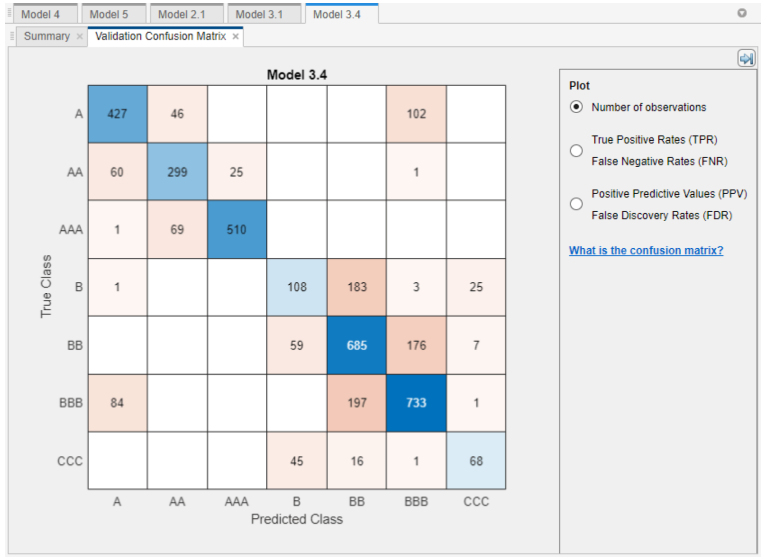
Practice #3: real-time sound processing



Machine learning with MATLAB Coder

Statistics and Machine Learning Toolbox


Train Model → Save Model → Define Entry-Point Function → Generate Code → Verify Generated Code



Model 3.4

True Class	A	AA	AAA	B	BB	BBB	CCC
A	427	46				102	
AA	60	299	25			1	
AAA	1	69	510				
B	1			108	183	3	25
BB				59	685	176	7
BBB	84				197	733	1
CCC				45	16	1	68

Predicted Class

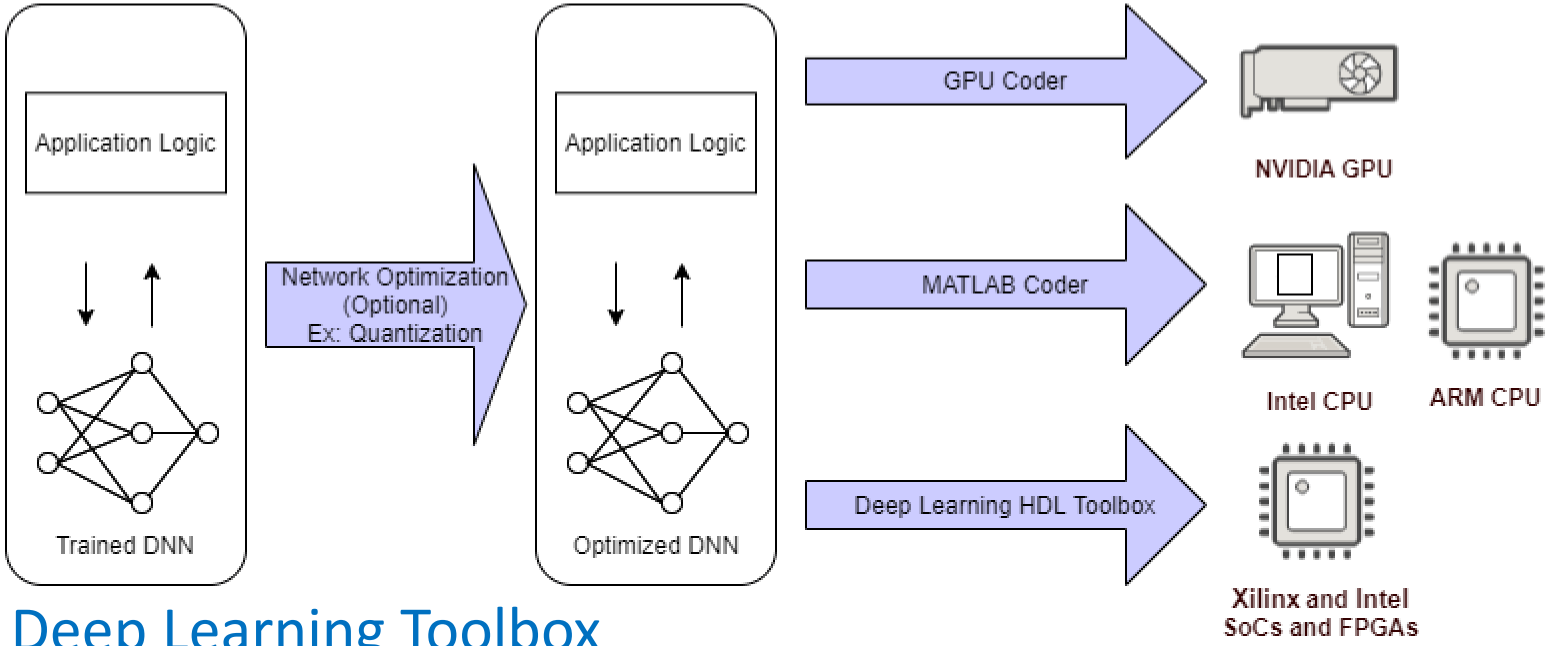
Plot

● Number of observations

○ True Positive Rates (TPR)
False Negative Rates (FNR)

○ Positive Predictive Values (PPV)
False Discovery Rates (FDR)

What is the confusion matrix?


Application Logic

Trained DNN

Network Optimization (Optional) Ex: Quantization

Application Logic

Optimized DNN

GPU Coder → NVIDIA GPU

MATLAB Coder → Intel CPU, ARM CPU

Deep Learning HDL Toolbox → Xilinx and Intel SoCs and FPGAs

Deep Learning Toolbox