M55M1 Smart Motion Detection

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code demonstrates how to use the motion detection feature in a low power mode on the M55M1 series microcontroller. Based on the object detection results, the system will dynamically enable or disable motion detection blocks to achieve optimal performance and energy efficiency.
BSP Version	M55M1_Series_BSP_CMSIS_V3.00.001
Hardware	NuMaker-M55M1(BETA) V1.1

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. Overview

This example code demonstrates how to use the low power mode in conjunction with the motion detection feature on the M55M1 series microcontroller. The core objective of this example is to intelligently control the activation or deactivation of motion detection blocks based on the results of object detection, thereby achieving energy savings.

Specifically, the system first uses object detection technology to analyze objects in the scene. Once a specific object is detected, the system decides whether to enable or disable the corresponding motion detection blocks based on predefined conditions. For example, if a human enters a certain area, the motion detection feature will be activated; conversely, if no human is detected, the system will disable the relevant blocks to save energy.

This method effectively optimizes the use of system resources, making it particularly suitable for applications that require long-term operation and are limited by battery capacity, such as smart surveillance and home security.

1.1 Principle

This example demonstrates how to use the M55M1 series microcontroller and the low power mode to achieve efficient motion detection. By combining object detection technology, the system can intelligently control the activation or deactivation of motion detection blocks, thereby achieving optimal performance and energy savings. This method has broad potential in smart device applications and is worth further research and promotion.

The overall process is shown in Figure 1-1.

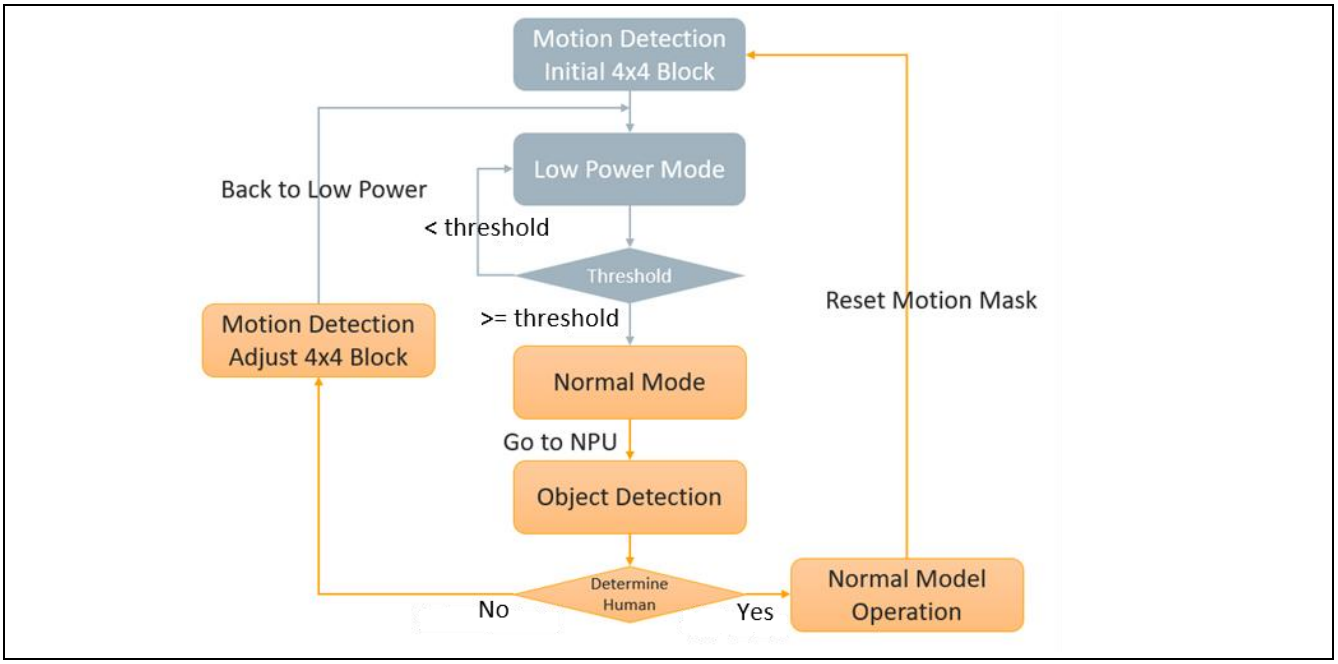

Figure 1-1 Smart Motion Detection Flowchart

1.2 Process Details

This system performs motion detection based on a power-saving mode, integrating the camera capture interface controller (CCAP) with the chip's functionality. When the system starts, the chip is in power-saving mode by default and only begins motion detection when triggered by a timer. First, the CCAP captures a 640x480 image in Y format, which is then scaled down to 320x240. Next, the image undergoes max pooling to further reduce its size to 160x120. The system then extracts a 20x15 sub-image from this, with a total of 16 windows. Based on whether the current frame is odd or even, the system decides whether to write the data into internal SRAM or read from it. If the frame is even, the system reads the window data from internal SRAM and calculates the sum of absolute differences (SAD) using an image matching algorithm to produce the final motion detection result. The system then checks the CCAP interrupt status. If an interrupt condition is met, the CPU is awakened, and an interrupt is triggered. If not, the system returns to power-saving mode to ensure energy efficiency.

When the detected motion energy is less than the set threshold, the system will keep in the low power mode to save energy. However, when the detected motion energy exceeds the threshold, the system will wake up to the normal mode and then do the object detection.

In normal mode, the Neural Processing Unit (NPU) performs object detection. Here, the YOLO (You Only Look Once) model, a fast and accurate object detection algorithm, is used. The trained YOLO model can determine whether the object in the image is a human or something else.

When the system determines that there is a human in the image, it enters user operation mode, allowing the user to perform related operations. On the other hand, when the system determines that there is no human in the image, the 4x4 blocks will be adjusted, the parts that caused false alarms will be disabled, and Motion Detection will return to the initial 4x4 block stage to restart the detection process. Until the system determines that there is a person in the image, the closed 4x4 block will be reopened and Motion Detection will return to its initial state.

This design effectively balances power consumption and detection accuracy, ensuring stable operation of the system in different environments.

- Motion Detection Initial Setup (Initial 4x4 Block of Motion Detection)

  The system first activates the motion detection feature and sets it to an initial 4x4 block size. This setup allows the system to perform preliminary detection in a small area to save power.

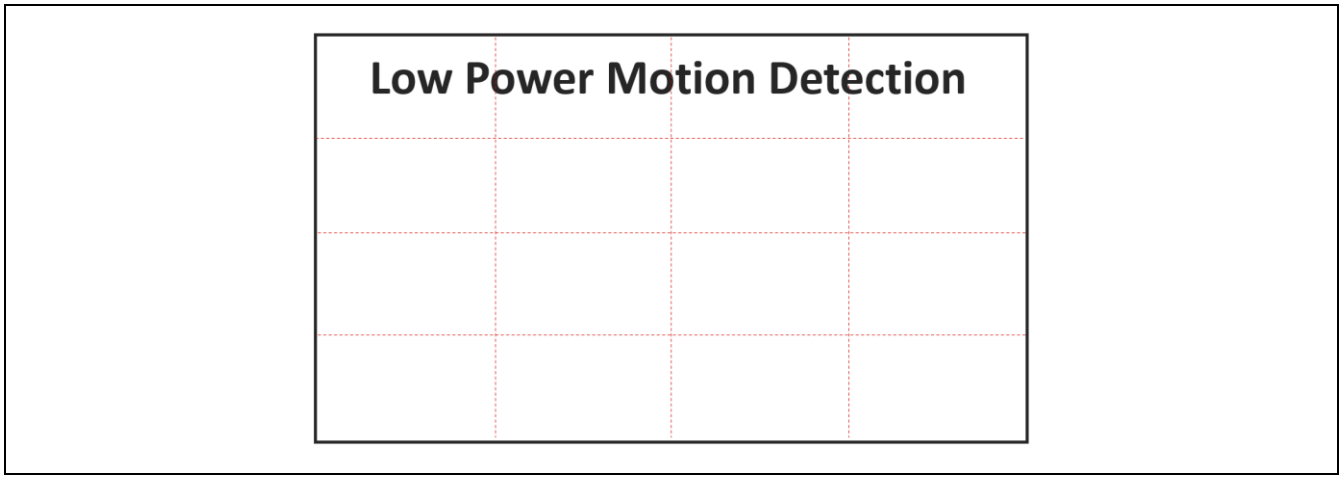

Figure 1-2 Initial 4x4 Block of Motion Detection

- Low Power Mode

  After the initial setup is complete, the system enters low power mode. In this mode, the motion detection feature operates with minimal power consumption to extend battery life and reduce energy consumption.

- Threshold

  The system evaluates the detected motion signals and compares them with the predefined threshold values.

  - If the detected motion signal strength is less than the threshold (< Th), the system will continue to operate in low power mode.
  - If the detected motion signal strength is greater than the threshold (> Th), the system will switch to normal mode and adjust the 4x4 block settings to adapt to new environmental conditions.

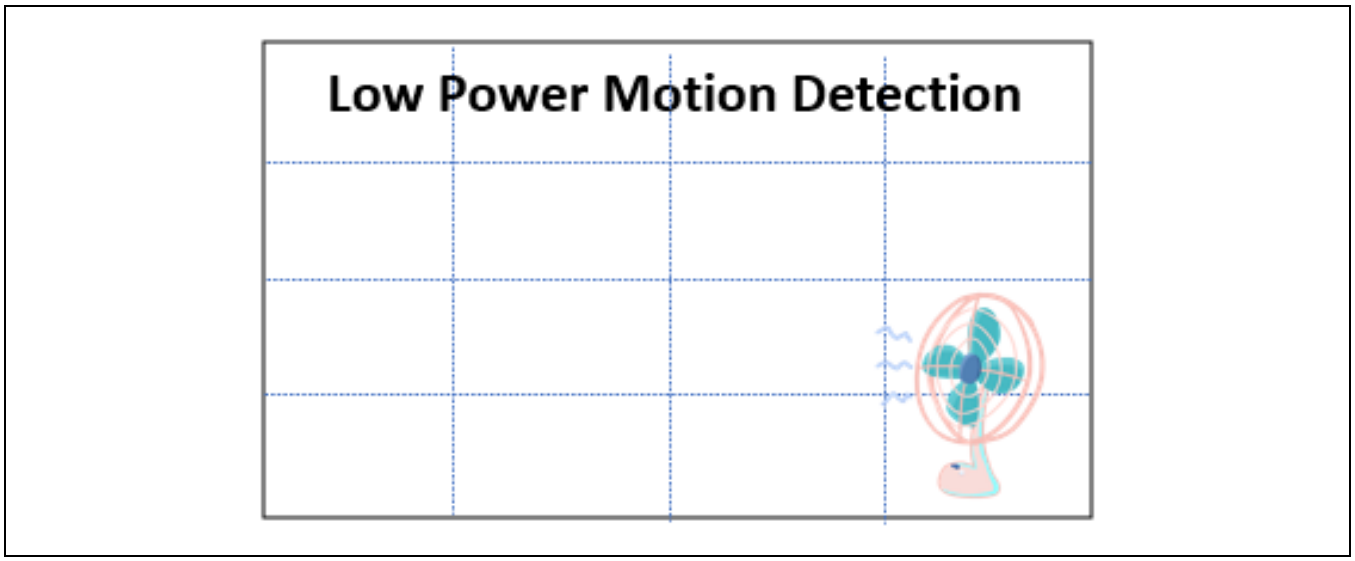

Figure 1-3 System Detects Motion

- Normal Mode

  When the signal strength exceeds the threshold, the system enters normal mode. In this mode, the motion detection feature is fully activated, allowing for more precise detection of changes in the scene.

- Object Detection

  In normal mode, the detected data is transmitted to the Neural Processing Unit (NPU) for object detection. The NPU uses deep learning algorithms to analyze the data and determine what the detected object is.

- Determine Human

  Based on the results of object detection, the system will determine whether the detected object is a human.

  If the detection result indicates a human (is human), the system will enter normal mode operation, continue executing the current tasks, and maintain the system settings.

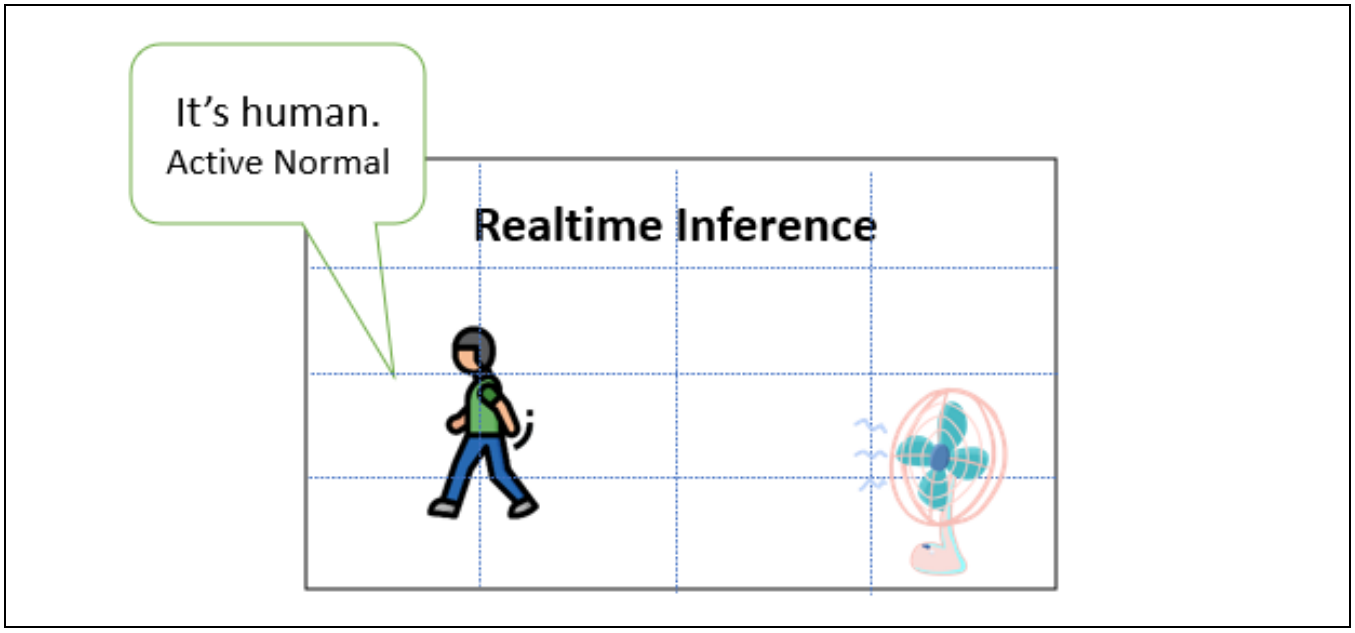

Figure 1-4 Detected Object is Human

If the detection result indicates it is not a human (isn't human), the system will return to adjusting the 4x4 block settings and may re-enter low power mode to continue the next round of detection.

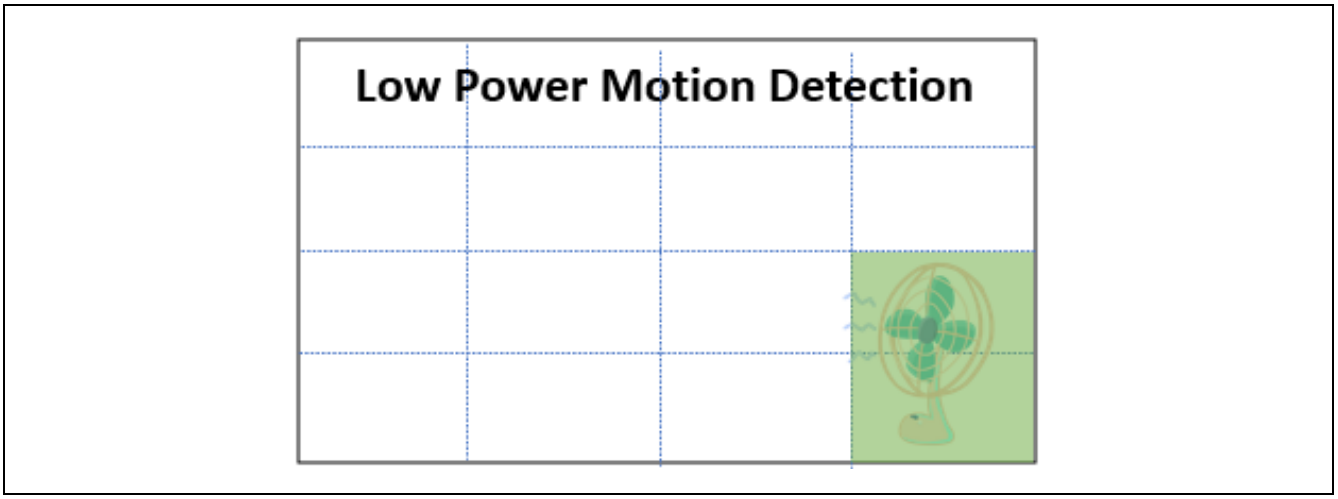

Figure 1-5 Detected Object is Not Human

1.3 Using Pre-Trained Model

When performing object detection, the YOLO (You Only Look Once) model is widely favored for its efficient inference speed and good accuracy. YOLO-Fastest-1.1 is a lightweight variant of the YOLO family, designed specifically for embedded devices and environments with limited computing resources. The following will detail the process and results of training and applying the YOLO-Fastest-1.1 model using the COCO dataset.

1.3.1 YOLO-Fastest-1.1 Model

YOLO-Fastest-1.1 is an improved version based on YOLOv3, designed to reduce computational resource requirements while maintaining a certain level of detection accuracy. The main features of this model are its small number of parameters and fast inference speed, making it suitable for real-time object detection on mobile devices and embedded systems. YOLO-Fastest-1.1 employs a more lightweight network architecture in its design, such as using ShuffleNetV2 as the backbone network, and introduces anchor matching mechanisms and loss functions from YOLOv5. This allows the model to reduce the number of parameters while maintaining speed.

1.3.2 COCO Dataset

The COCO (Common Objects in Context) dataset is a large-scale object detection, segmentation, and captioning dataset. It contains over 330,000 images, more than 200,000 of which are labeled, and includes over 80 object categories. The dataset is designed to spur progress in computer vision by providing diverse and complex scenes with objects in various contexts, making it a standard benchmark for evaluating the performance of object detection models. The COCO dataset's comprehensive annotations include not only object detection bounding boxes but also keypoints for human poses and object segmentation masks, making it a versatile resource for a wide range of computer vision tasks.

The COCO dataset includes the following sets:

- Training set: Contains approximately 118,000 images and their annotations.
- Validation set: Contains approximately 5,000 images and their annotations.
- Test set: Contains approximately 40,000 images, but none annotations.

1.3.3 Types of Labels

ID	Name	ID	Name	ID	Name	ID	Name
1	person	21	elephant	41	wine glass	61	dining table
2	bicycle	22	bear	42	cup	62	toilet
3	car	23	zebra	43	fork	63	tv
4	motorcycle	24	giraffe	44	knife	64	laptop
5	airplane	25	backpack	45	spoon	65	mouse
6	bus	26	umbrella	46	bowl	66	remote
7	train	27	handbag	47	banana	67	keyboard
8	truck	28	tie	48	apple	68	cell phone
9	boat	29	suitcase	49	sandwich	69	microwave
10	traffic light	30	frisbee	50	orange	70	oven
11	fire hydrant	31	skis	51	broccoli	71	toaster
12	stop sign	32	snowboard	52	carrot	72	sink
13	parking meter	33	sports ball	53	hot dog	73	refrigerator
14	bench	34	kite	54	pizza	74	book
15	bird	35	baseball bat	55	donut	75	clock
16	cat	36	baseball glove	56	cake	76	vase
17	dog	37	skateboard	57	chair	77	scissors
18	horse	38	surfboard	58	couch	78	teddy bear
19	sheep	39	tennis racket	59	potted plant	79	hair drier
20	cow	40	bottle	60	bed	80	toothbrush

1.3.4 Category for Person

The "person" category is one of the most common and important categories. This dataset aims to provide rich annotated data to support tasks such as object detection, semantic segmentation, and image captioning. The COCO dataset covers various everyday scenes, with each image accompanied by detailed annotation information, including object bounding boxes, semantic segmentation masks, and keypoints.

Bounding Box:

The bounding box annotates the position and size of each person in the image. These boxes are represented by four coordinate values (x, y, w, h), where (x, y) is the coordinate of the top-left corner of the bounding box, and w and h are the width and height of the bounding box, respectively.

For example, an annotation file for a "person" category might contain the following.

```
0 0.312 0.456 0.234 0.567
```

In this context, 0 is the category index, 0.312 and 0.456 are the normalized coordinates of the bounding box center, and 0.234 and 0.567 are the width and height of the bounding box.

1.4 Demo Result

The results of compiling and programming this example code are shown in Figure 1-6.

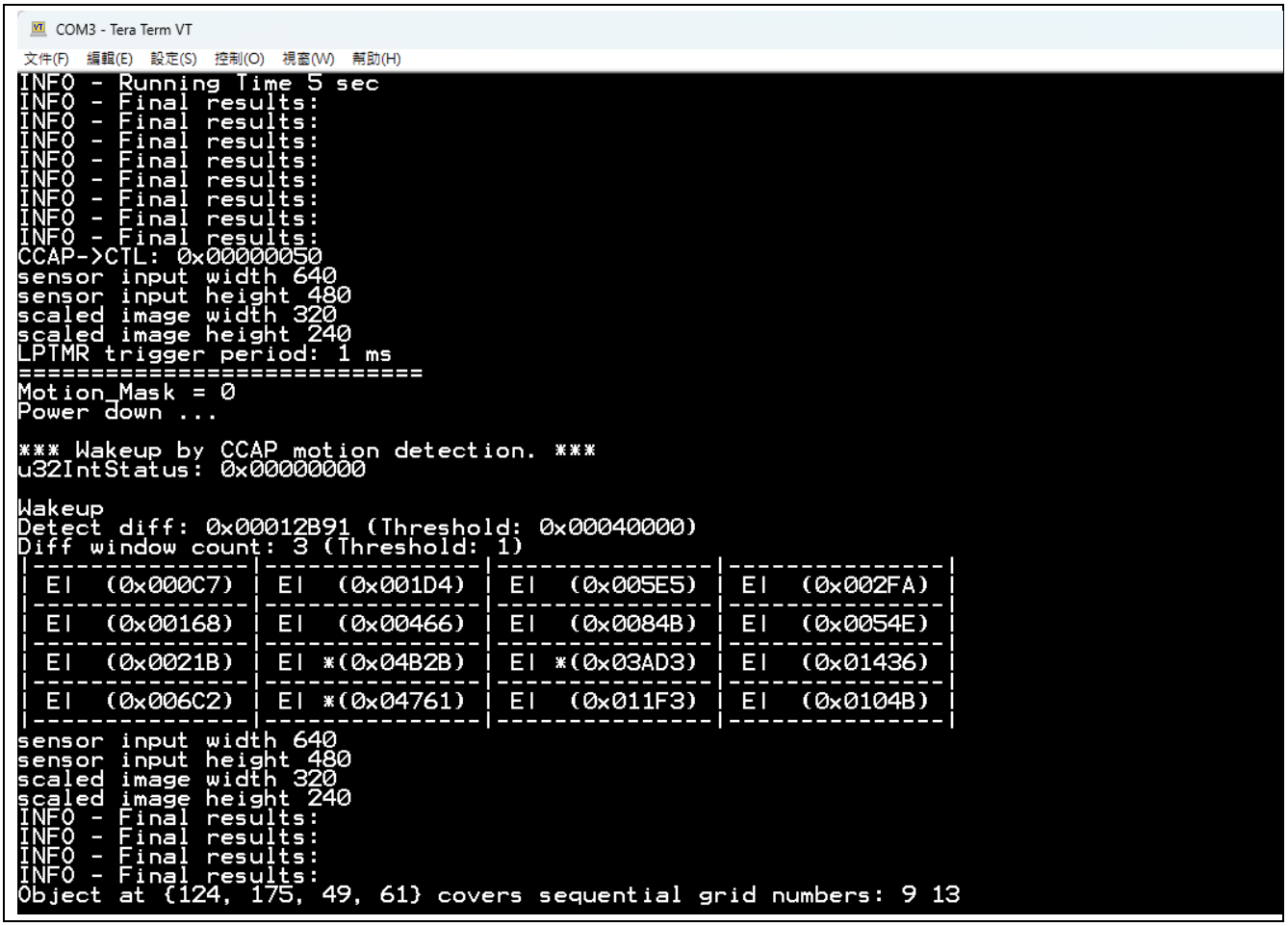

Figure 1-6 Execution of Compilation and Burning Results

The execution results will be printed out by the PC UART, showing the results of object detection and the detected motion blocks information. Below are the printed results of object detection, after which the system will return to Power-down mode.

```
INFO - BoardInit: complete
```

```
INFO - Ethos-U device initialised
INFO - Ethos-U version info:
INFO -  Arch:       v1.1.0
INFO -  Driver:     v0.16.0
INFO -  MACs/cc:    256
INFO -  Cmd stream: v0
INFO - Target system: M55M1
INFO - main task running
INFO - Set tesnor arena cache policy to WTRA
u32RegionCnt: 3, (MPU_REGIONS_MAX - i32RegionIdx): 7
INFO - Added ethos-u support to op resolver
INFO - Creating allocator using tensor arena at 0x82000000
INFO - Allocating tensors
INFO - Model INPUT tensors:
INFO -  tensor type is INT8
INFO -  tensor occupies 307200 bytes with dimensions
INFO -          0:   1
INFO -          1: 320
INFO -          2: 320
INFO -          3:   3
INFO - Quant dimension: 0
INFO - Scale[0] = 0.003922
INFO - ZeroPoint[0] = -128
INFO - Model OUTPUT tensors:
INFO -  tensor type is INT8
INFO -  tensor occupies 102000 bytes with dimensions
INFO -          0:   1
INFO -          1:  20
INFO -          2:  20
INFO -          3: 255
INFO - Quant dimension: 0
INFO - Scale[0] = 0.190095
INFO - ZeroPoint[0] = 77
INFO -  tensor type is INT8
INFO -  tensor occupies 25500 bytes with dimensions
INFO -          0:   1
INFO -          1:  10
INFO -          2:  10
INFO -          3: 255
INFO - Quant dimension: 0
INFO - Scale[0] = 0.220456
INFO - ZeroPoint[0] = 67
INFO - Activation buffer (a.k.a tensor arena) size used: 1229496
INFO - Number of operators: 3
INFO -  Operator 0: ethos-u
INFO -  Operator 1: RESIZE_NEAREST_NEIGHBOR
INFO -  Operator 2: ethos-u
INFO - init perfcycles 900001188
Sensor Chip_Version_H = 0x09 (0x09) Chip_Version_L = 0x55 (0x55)
sensor input width 640
sensor input height 480
scaled image width 320
scaled image height 240
INFO - Final results:
INFO - Final results:
INFO - Final results:
```

```
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Running Time 5 sec
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
INFO - Final results:
CCAP->CTL: 0x00000050
sensor input width 640
sensor input height 480
scaled image width 320
scaled image height 240
LPTMR trigger period: 1 ms
============================
Motion_Mask = 0
Power down ...
```

If there is a motion, the motion information will be printed as follows:

```
*** Wakeup by CCAP motion detection. ***
u32IntStatus: 0x00000000

Wakeup
Detect diff: 0x0000E3DB (Threshold: 0x00040000)
Diff window count: 3 (Threshold: 1)
|---------------|---------------|---------------|---------------|
| E|  (0x002CA) | E|  (0x00262) | E|  (0x00203) | E|  (0x001C8) |
|---------------|---------------|---------------|---------------|
| E|  (0x00203) | E|  (0x003F2) | E|  (0x00248) | E|  (0x00478) |
|---------------|---------------|---------------|---------------|
| E|  (0x009D9) | E|  (0x002FB) | E|  (0x005A2) | E|  (0x00AB7) |
|---------------|---------------|---------------|---------------|
| E| *(0x02A66) | E| *(0x0428C) | E|  (0x018A1) | E| *(0x02B6F) |
|---------------|---------------|---------------|---------------|
sensor input width 640
sensor input height 480
scaled image width 320
scaled image height 240
INFO - Final results:
Object at {68, 169, 61, 70} covers sequential grid numbers: 8 12
INFO - Final results:
```

If an object is detected, the information above will be printed, showing the object's position and label number:

Object at {68, 169, 61, 70} covers sequential grid numbers: 8 12

Because the threshold set in the program code is 0x2000 (you can refer to the motion detection settings introduced in Chapter 2 below), the activated window will be marked with a * symbol, as shown in the Motion information printed below.

```
|---------------|---------------|---------------|---------------|
| E|  (0x002CA) | E|  (0x00262) | E|  (0x00203) | E|  (0x001C8) |
|---------------|---------------|---------------|---------------|
| E|  (0x00203) | E|  (0x003F2) | E|  (0x00248) | E|  (0x00478) |
|---------------|---------------|---------------|---------------|
| E|  (0x009D9) | E|  (0x002FB) | E|  (0x005A2) | E|  (0x00AB7) |
|---------------|---------------|---------------|---------------|
| E| *(0x02A66) | E| *(0x0428C) | E|  (0x018A1) | E| *(0x02B6F) |
|---------------|---------------|---------------|---------------|
```

The numbering of grid numbers is as follows:

```
|---------------|---------------|---------------|---------------|
|       0       |       1       |       2       |       3       |
|---------------|---------------|---------------|---------------|
|       4       |       5       |       6       |       7       |
|---------------|---------------|---------------|---------------|
|       8       |       9       |      10       |      11       |
|---------------|---------------|---------------|---------------|
|      12       |      13       |      14       |      15       |
|---------------|---------------|---------------|---------------|
```

2. Code Description

Initialize the 4x4 block of motion detection, and the code is located in *main.cpp*.

```
//Power down and CCAP wakeup procedure
/* Config CCAP motion detection */
if (PacketMotionDetection(&g_sSensorHM1055_VGA_YUV422) != 0)
{
    printf("Test motion detection failed !\n");

}
ImageSensor_Config(eIMAGE_FMT_RGB565, CCAP_MD_WIDTH, CCAP_MD_HEIGHT);
/* Init CCAP clock and Sensor clock */
CLK_SetModuleClock(CCAP0_MODULE, CLK_CCAPSEL_CCAP0SEL_HIRC, 0);
/* Config LPTMR0 to trigger CCAP motion detection per 100 ms */
ConfigLPTMR(LPTMR0, 1);
CCAP_MD_SET_TRIG_SRC(CCAP_MD_TRIG_LPTMR0, TRUE);

printf("============================\n");
printf("Motion_Mask = %x \n", Motion_Mask);
CCAP_MD_ENABLE_WINDOW(0xFFFF);
CCAP_MD_DISABLE_WINDOW(Motion_Mask);

PMC_SetPowerDownMode(PMC_NPD0, PMC_PLCTL_PLSEL_PL1);
EnterPowerDown();

CLK_SetModuleClock(CCAP0_MODULE, CLK_CCAPSEL_CCAP0SEL_HXT, 0);
SPIM_HYPER_EnableDLL(HYPERRAM_SPIM_PORT);
ImageSensor_Config(eIMAGE_FMT_RGB565, frameBuffer.w, frameBuffer.h); //eIMAGE_FMT_RGB565

u64PowerDownCycle = (uint64_t)pmu_get_systick_Count() + (uint64_t)(SystemCoreClock *
POWERDOWN_SEC);
```

The following is a detailed explanation of how to use the Smart Motion Detection. The code implements real-time object detection and motion detection functions on an embedded system. Through detailed explanations of model initialization, image processing, motion detection setup, result display, and power-saving mode management, it demonstrates how to efficiently perform complex image processing tasks in resource-limited environments. The application of these technologies helps achieve low-power, high-performance embedded vision systems.

Model initialization:

```
/* Model Object Creation and Initialization */
arm::app::YoloFastestModel model;

if (!model.Init(arm::app::tensorArena,
                sizeof(arm::app::tensorArena),
                arm::app::yolofastest::GetModelPointer(),
                arm::app::yolofastest::GetModelLen())) {
    printf_err("Failed to initialise model\n");
    return;
}
```

```
TfLiteTensor *inputTensor = model.GetInputTensor(0);
TfLiteTensor *outputTensor = model.GetOutputTensor(0);

if (!inputTensor->dims) {
    printf_err("Invalid input tensor dims\n");
    return;
} else if (inputTensor->dims->size < 3) {
    printf_err("Input tensor dimension should be >= 3\n");
    return;
}

TfLiteIntArray *inputShape = model.GetInputShape(0);

const int inputImgCols = inputShape->data[arm::app::YoloFastestModel::ms_inputColsIdx];
const int inputImgRows = inputShape->data[arm::app::YoloFastestModel::ms_inputRowsIdx];
}
```

In this part, the code first creates and initializes the YOLO model object. The model.Init function initializes the model using the tensor arena, and obtains the model's pointer and length by calling GetModelPointer and GetModelLen. Next, the code retrieves the model's input tensor (inputTensor) and output tensor (outputTensor), and checks whether the dimensions of the input tensor are valid. If the input tensor's dimensions are less than three, an error message is output and the function returns.

Image Processing:

```
static void DrawImageDetectionBoxes(
    const std::vector<arm::app::object_detection::DetectionResult> &results,
    image_t *drawImg,
    std::vector<std::string> &labels) {
    for (const auto &result : results) {
        imlib_draw_rectangle(drawImg, result.m_x0, result.m_y0, result.m_w, result.m_h,
COLOR_B5_MAX, 1, false);
        imlib_draw_string(drawImg, result.m_x0, result.m_y0 - 16,
labels[result.m_cls].c_str(), COLOR_B5_MAX, 2, 0, 0, false,
                          false, false, false, 0, false, false);
    }
}
```

This function is responsible for drawing the detected objects' bounding boxes and labels on the image. The function receives the detection results, the image, and the list of labels, and draws the bounding box of each detected object on the image using the imlib_draw_rectangle function and the labels using the imlib_draw_string function.

Motion Detection Settings:

```
int32_t PacketMotionDetection(S_SENSOR_INFO *psSensorInfo) {
    int32_t i32RetCode = 0;
    int32_t i;

    /* Configure CCAPEN (CCAP_CTL[0]) to disable the capture interface */
    printf("CCAP->CTL: 0x%08X\n", CCAP->CTL);
    CCAP_Close();
    CCAP->CTL |= CCAP_CTL_VPRST_Msk;
```

```
    CLK_SysTickDelay(1000);
    CCAP->CTL &= ~CCAP_CTL_VPRST_Msk;

    /* Configure the control registers for the motion detection feature */
    CCAP_MD_SET_TOTAL_THRESHOLD(0x40000);

    /* Set the window motion detection threshold, with a maximum value of 0x12AD4 */
    for (i = 0; i < 16; i++)
        CCAP_MD_SET_WIN_THRESHOLD(i, 0x2000);

    /* Set the overflow window count threshold, with a maximum value of 15. */
    CCAP_MD_SET_OVERFLOW_WIN_THRESHOLD(1);

    /* Enable CCAP motion detection interrupt */
    CCAP_EnableInt(CCAP_INTSTS_MDINTF_MODE2_Msk);
    /* Enable external capture interrupt */
    NVIC_EnableIRQ(CCAP_IRQn);

    return i32RetCode;
}
```

This function is responsible for configuring the motion detection feature of CCAP. First, it disables the capture interface, then sets the global motion detection threshold and the motion detection threshold for each window. Next, it sets the overflow window count threshold and enables the CCAP motion detection interrupt and the external capture interrupt.

Result Display:

```
static bool PresentInferenceResult(const
std::vector<arm::app::object_detection::DetectionResult> &results,
                                   std::vector<std::string> &labels) {
    info("Final results:\n");
    for (const auto &result : results) {
        // If it is not a human
        if (strcmp(labels[result.m_cls].c_str(), "person") != 0) {
            int left = result.m_x0;
            int top = result.m_y0;
            int right = result.m_x0 + result.m_w;
            int bottom = result.m_y0 + result.m_h;

            // Ensure the boundary coordinates do not exceed the image dimensions
            left = std::max(0, std::min(left, 319));
            top = std::max(0, std::min(top, 239));
            right = std::max(0, std::min(right, 319));
            bottom = std::max(0, std::min(bottom, 239));

            std::set<int> coveredGrids;
            for (int y = top; y <= bottom; y += 60) { // Iterate within the height range
of the rectangle
                for (int x = left; x <= right; x += 80) { // Iterate within the width
range of the rectangle
                    auto grid_pos = getGridPosition(x, y);
                    coveredGrids.insert(getSequentialNumber(grid_pos.first,
grid_pos.second));
                }
            }
```

```
            // Output all the grid sequence numbers covered by the rectangle
            printf("Object at {%d, %d, %d, %d} covers sequential grid numbers: ", left,
top, right - left, bottom - top);
            for (int num : coveredGrids) {
                printf("%d ", num);
                switch(num) {
                    case 0: Motion_Mask = Motion_Mask | 0x0001; break;
                    case 1: Motion_Mask = Motion_Mask | 0x0002; break;
                    case 2: Motion_Mask = Motion_Mask | 0x0004; break;
                    case 3: Motion_Mask = Motion_Mask | 0x0008; break;
                    case 4: Motion_Mask = Motion_Mask | 0x0010; break;
                    case 5: Motion_Mask = Motion_Mask | 0x0020; break;
                    case 6: Motion_Mask = Motion_Mask | 0x0040; break;
                    case 7: Motion_Mask = Motion_Mask | 0x0080; break;
                    case 8: Motion_Mask = Motion_Mask | 0x0100; break;
                    case 9: Motion_Mask = Motion_Mask | 0x0200; break;
                    case 10: Motion_Mask = Motion_Mask | 0x0400; break;
                    case 11: Motion_Mask = Motion_Mask | 0x0800; break;
                    case 12: Motion_Mask = Motion_Mask | 0x1000; break;
                    case 13: Motion_Mask = Motion_Mask | 0x2000; break;
                    case 14: Motion_Mask = Motion_Mask | 0x4000; break;
                    case 15: Motion_Mask = Motion_Mask | 0x8000; break;
                }
            }
            printf("\n");
        } else {
            // If it is a human.
            CCAP_MD_ENABLE_WINDOW(0xFFFF);
            Motion_Mask = 0x0;
        }
    }
    return true;
}
```

This function processes the inference results and updates the Motion_Mask based on the results. If the detected object is not a human, it calculates the covered grids and updates the Motion_Mask accordingly; if the detected object is a human, it resets the Motion_Mask.

Power Saving Mode Management:

```
static void EnterPowerDown(void) {
    printf("Power down ...\n");
    UART_WAIT_TX_EMPTY(DEBUG_PORT);

    /* Switch SCLK to HIRC when entering power-saving mode */
    CLK_SetBusClock(CLK_SCLKSEL_SCLKSEL_HIRC, CLK_APLLCTL_APLLSRC_HIRC, 0);
    /* Enter power down mode */
    PMC_PowerDown();

    /* Set the processor to use sleep as its low power mode */
    SCB->SCR &= ~SCB_SCR_SLEEPDEEP_Msk;

    /* Enable PLL0 180MHz clock from HIRC and switch the SCLK clock source to PLL0 */
    CLK_SetBusClock(CLK_SCLKSEL_SCLKSEL_APLL0, CLK_APLLCTL_APLLSRC_HIRC, FREQ_180MHZ);
    printf("\nWakeup\n");
```

```
#if 1   /* Check frame differences */
    int32_t i;
    printf("Detect diff: 0x%08X (Threshold: 0x%08X)\n", CCAP->MDTSAD, CCAP->MDTTH);
    printf("Diff window count: %d (Threshold: %d)\n", CCAP_MD_GET_OVERFLOW_WIN_CNT(),
CCAP->MDWOCTH);

    for (i = 0; i < CCAP_MD_WINDOW_CNT; i++) {
        if (i % 4 == 0) {
            printf("|---------------|---------------|---------------|---------------|\n");
            printf("|");
        }

        if (CCAP_MD_GET_WIN_SAD(i) > CCAP->MDWTH[i])
            printf(" %s| *(0x%05X) |", (CCAP->MDCTL & (1 << i)) ? "E" : "D",
CCAP_MD_GET_WIN_SAD(i));
        else
            printf(" %s|  (0x%05X) |", (CCAP->MDCTL & (1 << i)) ? "E" : "D",
CCAP_MD_GET_WIN_SAD(i));

        if ((i + 1) % 4 == 0)
            printf("\n");
    }

    printf("|---------------|---------------|---------------|---------------|\n");
#endif
}
```

This function manages the power-saving mode of the embedded system. It first switches the system clock to the High-Frequency Internal Oscillator (HIRC) and then enters power-saving mode. Upon waking up, the system reconfigures the clock and checks the frame differences to ensure the motion detection feature is functioning correctly.

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M55M1_Series_BSP_CMSIS_V3.00.001
- IDE version
  - Keil uVersion 5.38

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M55M1(BETA) V1.1
- Pin Connect
  - Connect the UART6 TX (PH.4) pin to the PC UART RX to show the execution results of this example code.

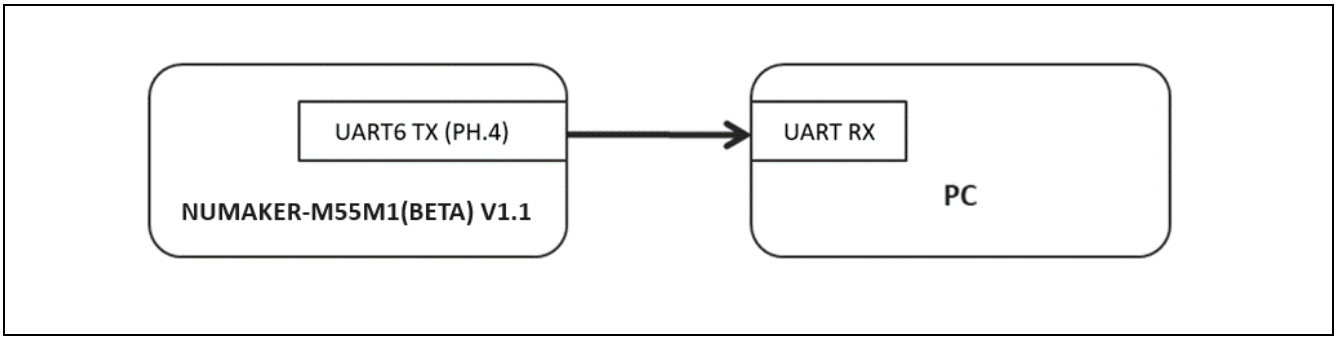

Figure 3-1 Pin Connect

  - The M55M1 uses I^{2}C to transmit control commands to the CMOS sensor and uses CCAP_SCLK as the SCLK to supply frequency to the CMOS sensor. The CMOS sensor returns the Pixel Clock (PIXCLK), HSYNC, VSYNC, and D0~D7 data according to the settings.

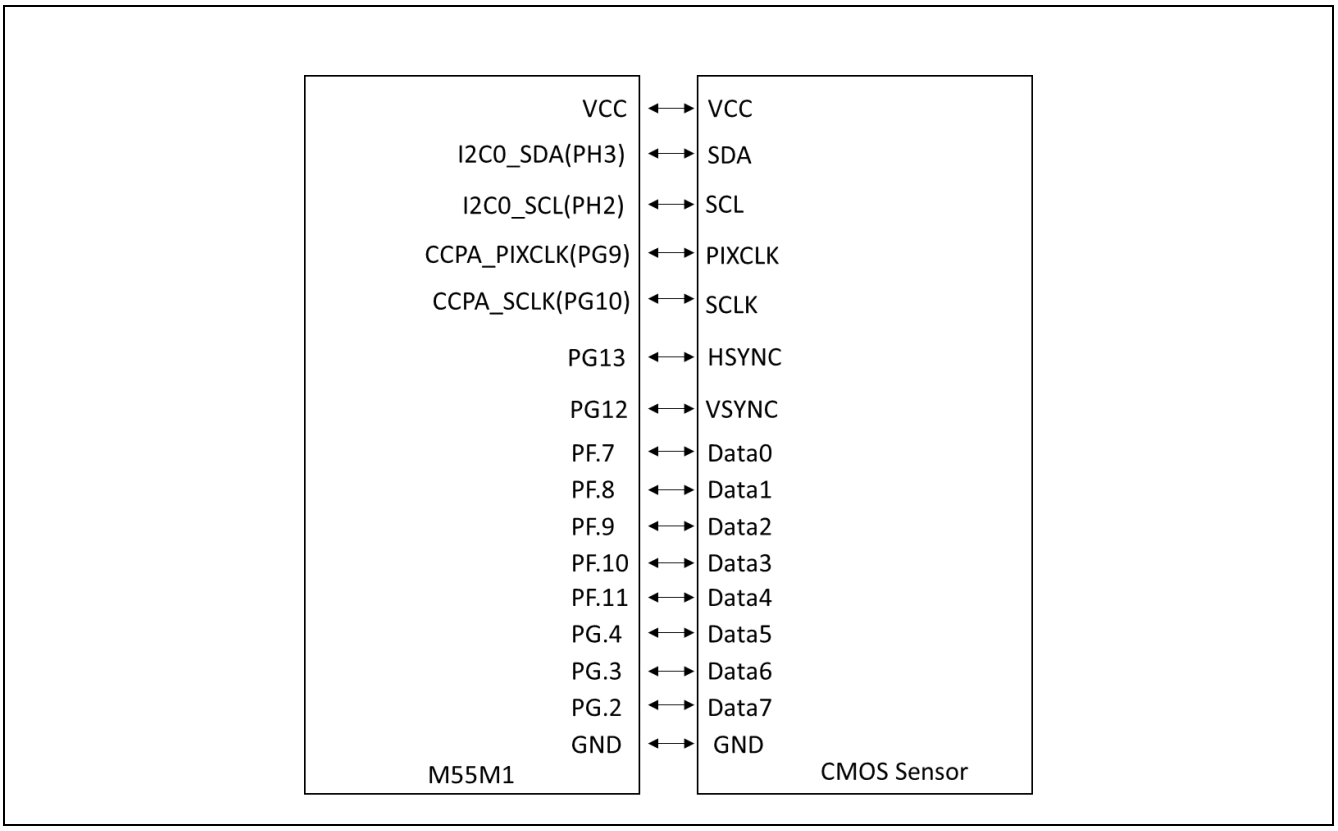

Figure 3-2 CMOS Circuit Diagram

4. Directory Information

The directory structure is shown below.

EC_M55M1_Smart_Motion_Detection_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
MachineLearning	
Project	Source file of example code

Figure 4-1 Directory Structure

5. Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *M55M1_Smart_Motion_Detection.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2024.06.24	1.00	Initial version.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.