M55M1 Smart Motion Detection

NuMicro® 32 位元系列微控制器範例程式碼介紹

文件資訊

應用簡述	本範例程式碼展示如何在 M55M1 上結合使用低功耗模式下的動作偵測功能。根據物件偵測結果，系統將動態地啟用或屏蔽動作偵測區塊，以達到最佳效能和節能效果。
BSP 版本	M55M1_Series_BSP_CMSIS_V3.00.001
開發平台	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. 概述

在本範例程式碼中，在此展示了如何在 M55M1 微控制器上結合使用低功耗模式（Low Power Mode）和動作偵測功能（Motion Detection Feature）。此範例的核心目標是根據人的物件偵測（Object Detection）結果，智慧地控制動作偵測區塊的開啟或屏蔽，從而達到節能效果。

具體來說，系統首先利用物件偵測技術來分析場景中的物件。一旦偵測到特定物件，系統會根據預先設定的條件決定是否啟用或屏蔽對應的動作偵測區塊。例如，如果人進入某個區域，動作偵測功能將被激活；反之，如果沒有偵測到人，系統會屏蔽相關區塊以節省電能。

這種方法能夠有效地最佳化系統資源的使用，特別適合於需要長時間運行且受限於電池容量的應用場景，如智慧監控、家居安防等。

1.1 原理

本範例展示了如何利用 M55M1 微控制器和低功耗模式來實現高效的動作偵測功能。通過結合物件偵測技術，系統能夠智慧地控制動作偵測區塊的開啟或屏蔽，從而達到最佳效能和節能效果。這種方法在智慧設備應用中具有廣泛的潛力，值得進一步研究和推廣。

整體流程如所示圖 1-1。

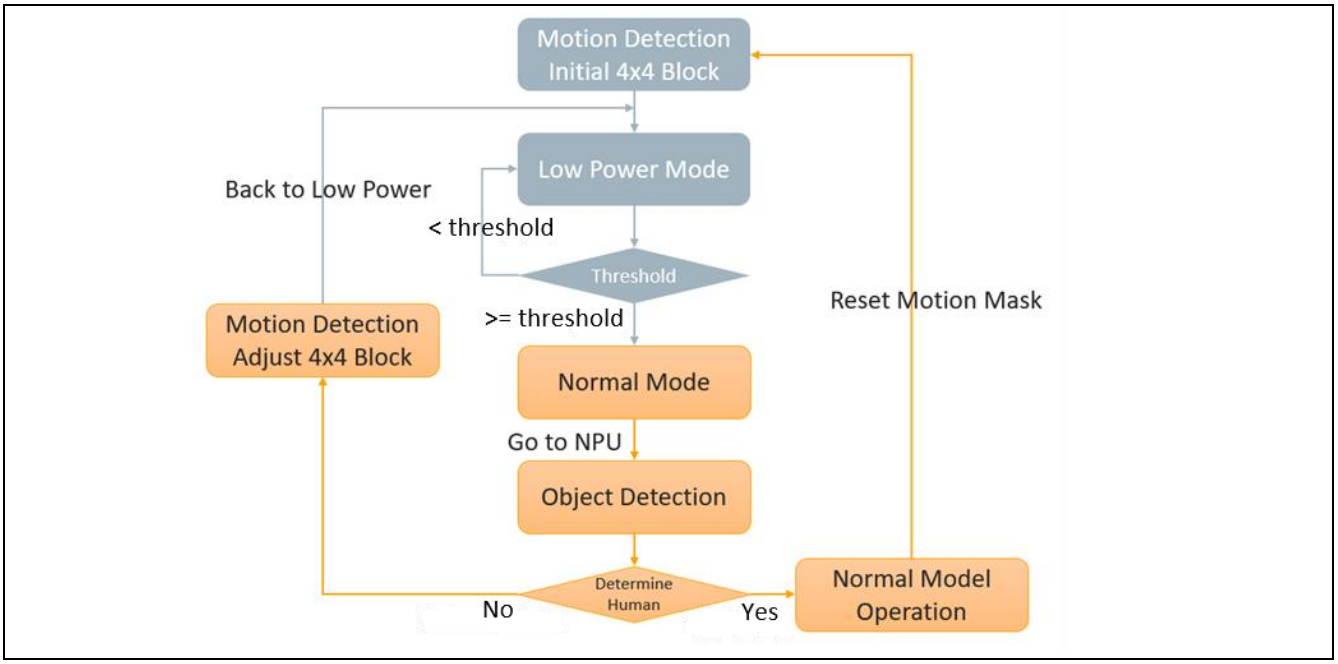

圖 1-1 Smart Motion Detection 流程圖

1.2 流程詳述

這個系統基於省電模式進行動作偵測，整合了相機擷取控制器（CCAP）與晶片的功能。在系統啟動時，晶片預設處於省電模式，只有在定時器觸發後，才開始進行動作偵測。首先，CCAP 擷取一張 640x480 的 Y 格式影像，並縮小至 320x240 的影像。接著，透過最大池化處理 (Max Pooling) 將影像進一步縮小為 160x120。然後，系統從該影像中提取 20x15 的子影像，共 16 個窗口。通過影像比對算法計算絕對差值總和 (SAD)，來生成最終的動作偵測結果。系統隨後檢查 CCAP 中斷狀態，若中斷條件成立，則喚醒 CPU 並觸發中斷；若不成立，則系統返回省電模式，以確保能源效率。

當檢測到的運動能量小於設定的閾值時，系統會保持在低功耗模式（Low Power Mode），以節省能源。然而，當檢測到的運動能量大於閾值時，系統會 Wake up 進入正常模式（Normal Mode）並啟動 NPU 做目標檢測。

在正常模式下，神經網絡處理單元（NPU）會進行目標檢測。這裡使用的是 YOLO 模型（You Only Look Once），這是一種快速而準確的目標檢測算法。訓練好的 YOLO 模型可以判斷圖像中的對象是人還是其他物件。

當系統判斷圖像中存在人時，會進入用戶操作模式，允許用戶進行相關操作。另一方面，當系統判斷圖像中不存在人時，會進行 4x4 區塊的調整，將誤報（False Alarm）的區塊關閉，並回到 Motion Detection 的初始 4x4 區塊階段，重新開始檢測過程，一直到系統判斷圖像中存在人時，被關閉的 4x4 區塊會被重新打開，Motion Detection 會回到初始的狀態。

這樣的設計可以有效地平衡功耗和檢測精度，確保系統在不同環境下的穩定運行。

- 動作偵測初始設置（Motion Detection Initial 4x4 Block）

  系統首先啟動動作偵測功能，並將其設置為初始的 4x4 區塊大小。這個設置允許系統在一個小範圍內進行初步偵測，以便節省電力。

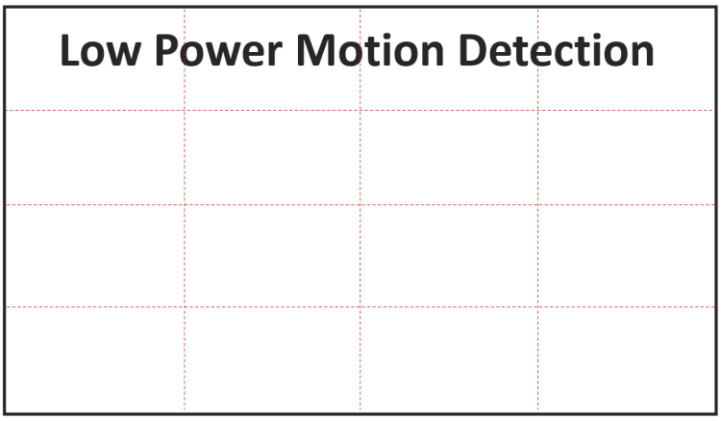

圖 1-2 Motion Detection Initial 4x4 Block

- 低功耗模式（Low Power Mode）

  在初始設置完成後，系統進入低功耗模式。在這種模式下，動作偵測功能以最低的能耗運行，目的是延長電池壽命和減少能源消耗。

- 門檻值判斷（Threshold）

  系統對偵測到的動作信號進行評估，並與預設的門檻值進行比較：

  - 如果偵測到的動作信號強度小於門檻值（< Th），系統將繼續運行在低功耗模式。
  - 如果偵測到的動作信號強度大於門檻值（> Th），系統將切換到正常模式進行更高級別的偵測，並調整 4x4 區塊的設置，以適應新的環境條件。

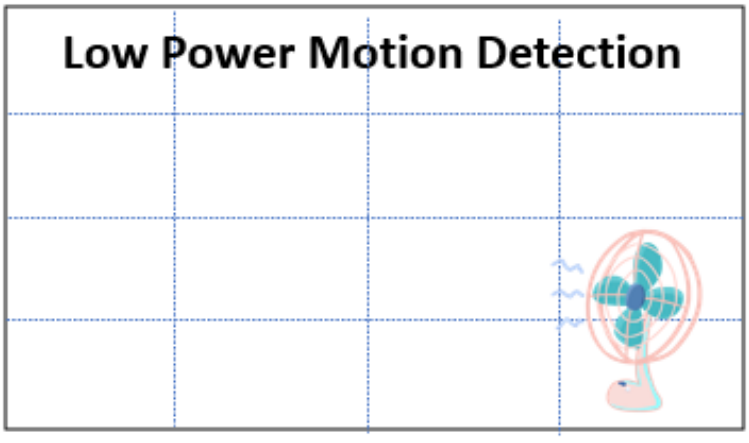

圖 1-3 當系統有偵測到 Motion 的情況

- 正常模式（Normal Mode）

  當信號強度超過門檻值時，系統進入正常模式。在此模式下，動作偵測功能全面啟動，能夠更精確地偵測到場景中的變化。

- 物件偵測（Object Detection）

  在正常模式下，偵測到的數據將被傳送到 NPU（神經處理單元）進行物件偵測。NPU 利用深度學習算法來分析數據，並確定偵測到的物件是什麼。

- 判斷是否為人類（Determine Human）

  根據物件偵測的結果，系統將判斷偵測到的物件是否為人類：

  如果偵測結果顯示為人類（is human），系統將進入正常模式操作（Normal Mode Operation），繼續執行當前任務並保持系統設置。

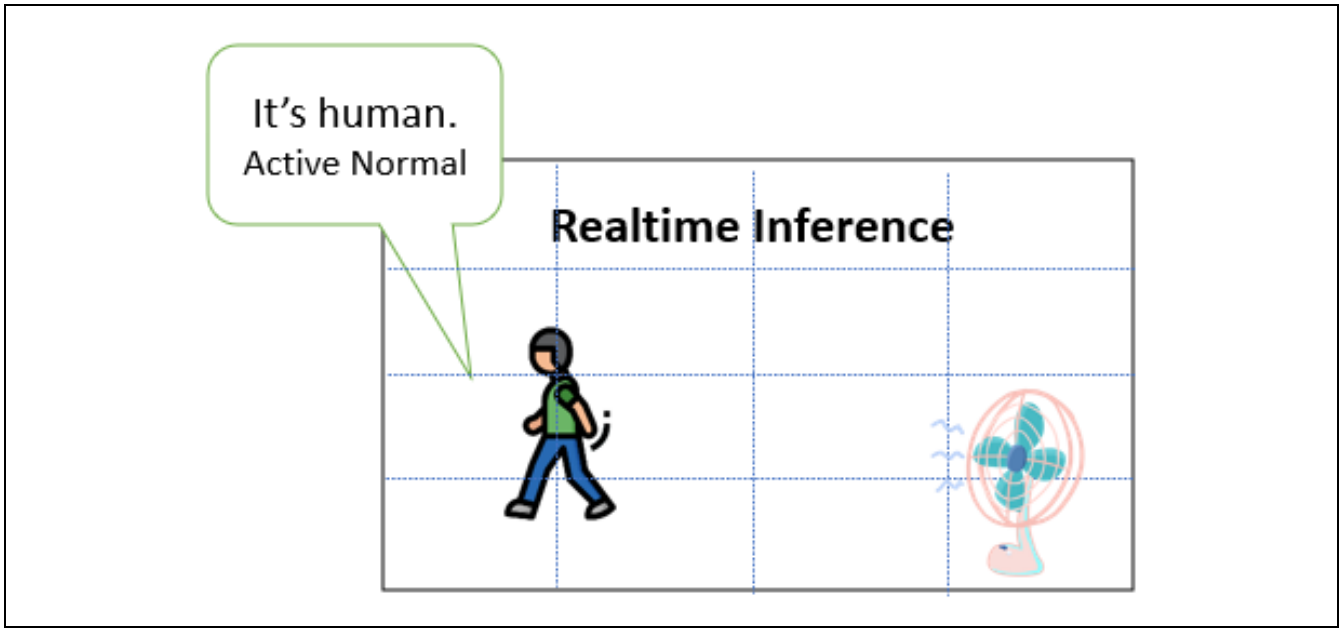

圖 1-4 偵測到的物件是人

如果檢測結果顯示不是人類（is not human），系統將返回調整 4x4 塊設置，並重新進入低功耗模式以繼續下一輪檢測。

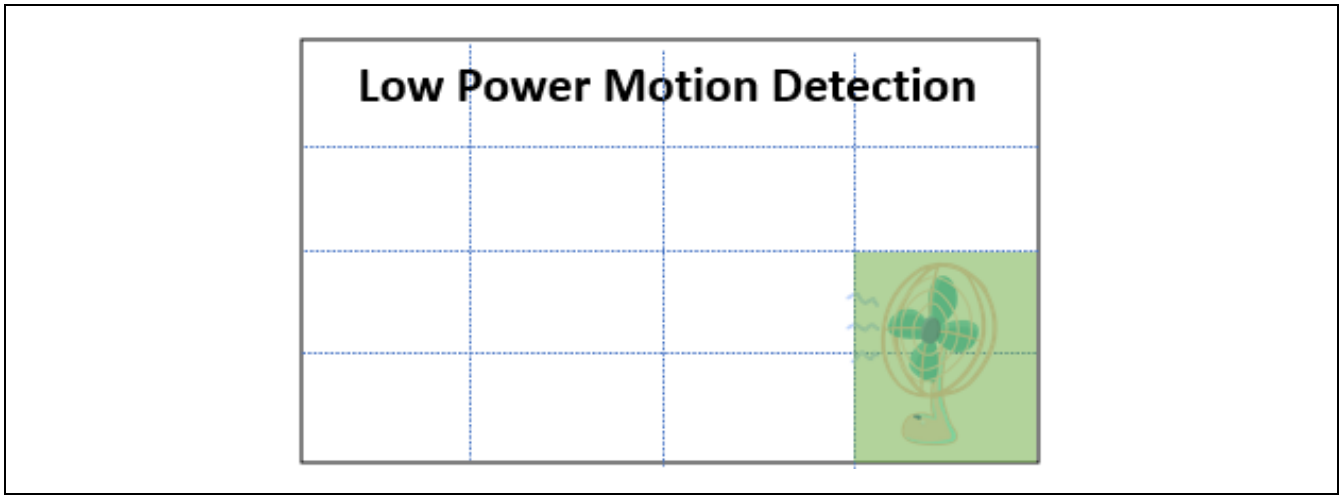

圖 1-5 系統屏蔽掉不是人的區塊

1.3 使用 Pre-Trained Model

在進行目標檢測時，YOLO（You Only Look Once）模型因其高效的推理速度和良好的精度而廣受歡迎。YOLO-Fastest-1.1 是 YOLO 家族中的一個輕量級變體，專為嵌入式設備和計算資源有限的環境設計。本文將詳細描述使用 YOLO-Fastest-1.1 模型及 COCO 資料集進行訓練和應用的過程和結果。

1.3.1 YOLO-Fastest-1.1 模型簡介

YOLO-Fastest-1.1 是基於 YOLOv3 的改進版本，旨在降低計算資源的需求，同時保持一定的檢測精度。這個模型的主要特點是參數量小、推理速度快，適合在移動設備和嵌入式系統上進行即時目標檢測。YOLO-Fastest-1.1 在設計上採用了更輕量的網絡架構，如使用 shufflenetV2 作為骨幹網，並引入了 YOLOv5 的錨點匹配機制和損失函數，使得模型能在保持速度的同時減少參數量。

1.3.2 COCO 資料集簡介

COCO（Common Objects in Context）資料集是目標檢測和圖像分割領域中最常用的資料集之一。它包含了 80 個類別的物件，涵蓋了日常生活中常見的各種場景和物品。每張圖像都附有豐富的標註信息，包括物件的邊界框、語義分割掩碼和關鍵點等。這使得 COCO 資料集成為訓練和評估目標檢測模型的理想選擇。

COCO 資料集包含以下內容：

- 訓練集：包含約 118,000 張圖像及其標註。
- 驗證集：包含約 5,000 張圖像及其標註。
- 測試集：包含約 40,000 張圖像，但沒有標註。

1.3.3 Label 的種類

編號	類別名稱	編號	類別名稱	編號	類別名稱	編號	類別名稱
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 Person 的類別

"person" 類別是最常見和最重要的類別之一。該資料集旨在提供豐富的標註數據以支持目標檢測、語義分割和圖像標註等任務。COCO 資料集涵蓋了日常生活中的各種場景，並且每張圖像都附有詳細的標註信息，其中包括物件的邊界框、語義分割掩碼和關鍵點等。

邊界框（Bounding Box）：

邊界框標註了每個人在圖像中的位置和大小。這些框是以四個座標值（x, y, w, h）來表示的，其中 (x, y) 是邊界框左上角的座標，w 和 h 分別是邊界框的寬度和高度。

例如，一個 “person” 類別的標註文件可能包含以下內容：

```
0 0.312 0.456 0.234 0.567
```

其中，0 是類別索引，0.312 和 0.456 是邊界框中心的歸一化座標，0.234 和 0.567 是邊界框的寬度和高度。

1.4 執行結果

本範例程式碼執行編譯和燒錄的結果如圖 1-6 所示。

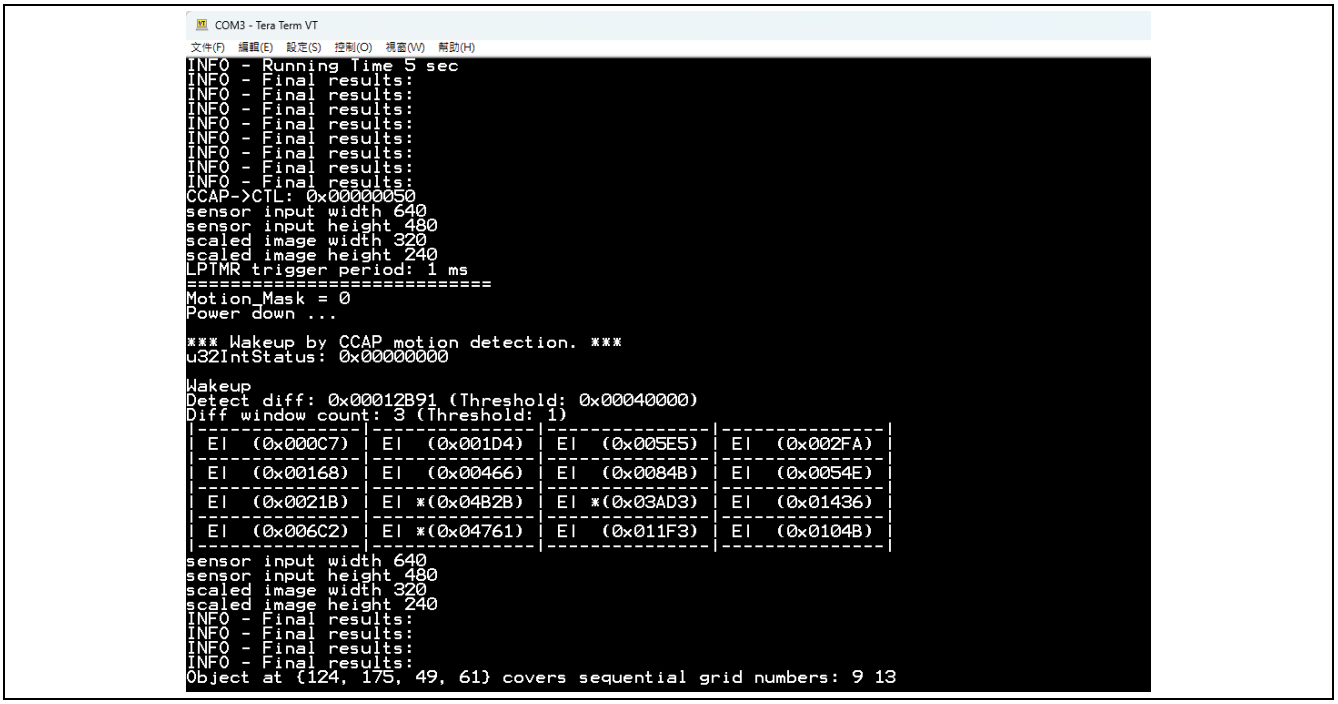

圖 1-6 執行編譯和燒錄結果

執行結果會由 PC UART 輸出出 Object Detection 的結果，以及偵測到 Motion 區塊等資訊。

以下為 Object Detection 輸出結果，結束後會回到 Power Down 的模式。

```
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 - 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 ...
```

如果有 Motion，會輸出 Motion 的資訊如下:

```
*** 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:
```

如果有偵測到物件，就會如上資訊，列印出 Object 的位置，與 Label 的號碼:

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

因為程式碼設定的 Threshold 是 0x2000（可以參考以下第 2 章程式碼介紹的運動檢測設定）被 Activate 的 window，會被標註 * 符號，如下輸出 Motion 的資訊所示。

```
|---------------|---------------|---------------|---------------|
| 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) |
|---------------|---------------|---------------|---------------|
```

關於 grid numbers 的編號，如下所示:

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

2. 程式碼介紹

初始化 motion detection 的 4x4 Block，程式碼位於 *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);
```

以下是詳細講解實際使用 Smart Motion Detection 的方法。

程式碼實現了嵌入式系統上的即時物件偵測和運動檢測功能。通過對模型初始化、影像處理、運動檢測設置、結果顯示和省電模式管理的詳細解說，展示了如何在資源有限的環境中有效地執行複雜的影像處理任務。這些技術的應用有助於實現低功耗、高效能的嵌入式視覺系統。

模型初始化：

```
/* 模型對象創建和初始化 */
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];
}
```

在這一部分，程式碼首先創建並初始化 YOLO 模型對象。model.Init 函數使用張量緩衝區（tensorArena）來初始化模型，並通過調用 GetModelPointer 和 GetModelLen 來獲取模型的指標和長度。接著，程式碼獲取模型的輸入張量（inputTensor）和輸出張量（outputTensor），並檢查輸入張量的維度是否有效。如果輸入張量的維度小於三，則輸出錯誤信息並返回。

影像處理：

```
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);
    }
}
```

此函數負責在影像上繪製檢測到的物件邊界框和標籤。函數接收檢測結果、影像和標籤列表，並在影像上繪製每個檢測到的物件的邊界框（使用 imlib_draw_rectangle 函數）和標籤（使用 imlib_draw_string 函數）。

運動檢測設定：

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

    /* 配置 CCAPEN(CCAP_CTL[0]) 以禁用捕獲介面 */
    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;

    /* 配置運動檢測功能的控制暫存器 */
    CCAP_MD_SET_TOTAL_THRESHOLD(0x40000);

    /* 設置窗口運動檢測閾值，最大值為 0x12AD4 */
    for (i = 0; i < 16; i++)
        CCAP_MD_SET_WIN_THRESHOLD(i, 0x2000);

    /* 設置溢出窗口計數閾值，最大值為 15。 */
    CCAP_MD_SET_OVERFLOW_WIN_THRESHOLD(1); // 8

    /* 啟用 CCAP 運動檢測中斷 */
    CCAP_EnableInt(CCAP_INTSTS_MDINTF_MODE2_Msk);
    /* 啟用外部捕獲中斷 */
    NVIC_EnableIRQ(CCAP_IRQn);

    return i32RetCode;
}
```

此函數負責配置 CCAP 的運動檢測功能。首先，禁用捕獲介面，然後設置全局運動檢測閾值和每個窗口的運動檢測閾值。接著，設置溢出窗口計數閾值，並啟用 CCAP 運動檢測中斷和外部捕獲中斷。

結果顯示：

```
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) {
        // 如果不是人
        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;

            // 確保邊界座標不超出影像尺寸
            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) { // 在矩形高度範圍內迭代
                for (int x = left; x <= right; x += 80) { // 在矩形寬度範圍內迭代
                    auto grid_pos = getGridPosition(x, y);
                    coveredGrids.insert(getSequentialNumber(grid_pos.first,
grid_pos.second));
                }
            }

            // 輸出所有被矩形覆蓋的網格序列號
            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 {
            // 如果是人
            CCAP_MD_ENABLE_WINDOW(0xFFFF);
            Motion_Mask = 0x0;
        }
    }
    return true;
}
```

此函數用於處理推論結果，並根據結果更新 Motion_Mask。如果檢測到的物件不是人，則計算其覆蓋的網格並更新 Motion_Mask；如果檢測到的是人，則重置 Motion_Mask。

省電模式管理：

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

```
    /* 進入省電模式時切換 SCLK 到 HIRC */
    CLK_SetBusClock(CLK_SCLKSEL_SCLKSEL_HIRC, CLK_APLLCTL_APLLSRC_HIRC, 0);
    /* 進入省電模式 */
    PMC_PowerDown();

    /* 設置處理器使用睡眠作為其低功耗模式 */
    SCB->SCR &= ~SCB_SCR_SLEEPDEEP_Msk;

    /* 啟用 PLL0 180MHz 時鐘從 HIRC 並切換 SCLK 時鐘源到 PLL0 */
    CLK_SetBusClock(CLK_SCLKSEL_SCLKSEL_APLL0, CLK_APLLCTL_APLLSRC_HIRC, FREQ_180MHZ);
    printf("\nWakeup\n");

#if 1   /* 檢查幀差異 */
    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
}
```

此函數管理嵌入式系統的省電模式。它首先切換系統時鐘到高頻內部振盪器（HIRC），然後進入省電模式。喚醒後，系統重新配置時鐘，並檢查幀差異以確保運動檢測功能正常。

3. 軟體與硬體需求

3.1 軟體需求

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

3.2 硬體需求

- 電路元件
  - NuMaker-M55M1(BETA) V1.1
- 線路示意圖
  - 將 UART6 TX (PH.4) pin 腳連接到 PC UART RX，以顯示範例程式碼的執行結果。

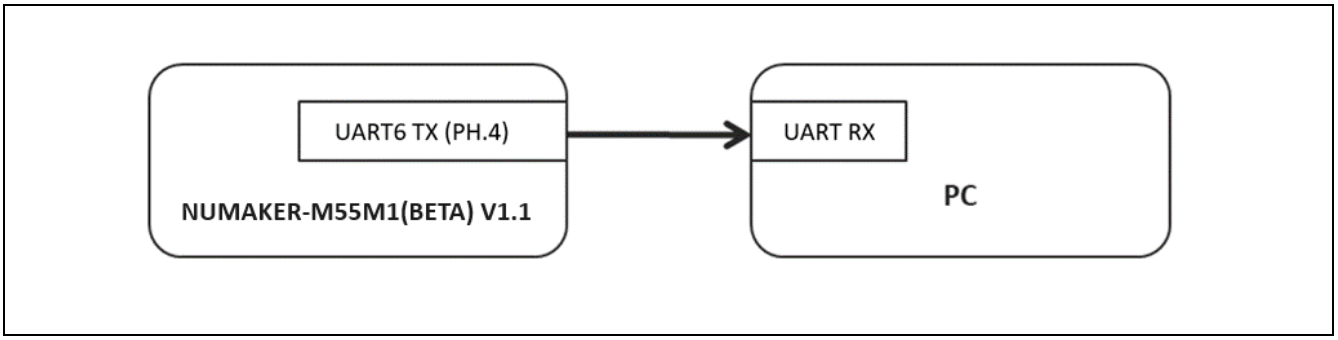

圖 3-1 線路示意圖

  - M55M1 使用 I2C 來傳輸控制命令至 CMOS 感測器，使用 CCAP_SCLK 作為 SCLK 供應頻率給 CMOS 感測器，CMOS 感測器會根據設定回傳 Pixel Clock (PIXCLK)、HSYNC、VSYNC 和 D0~D7 資料數據。

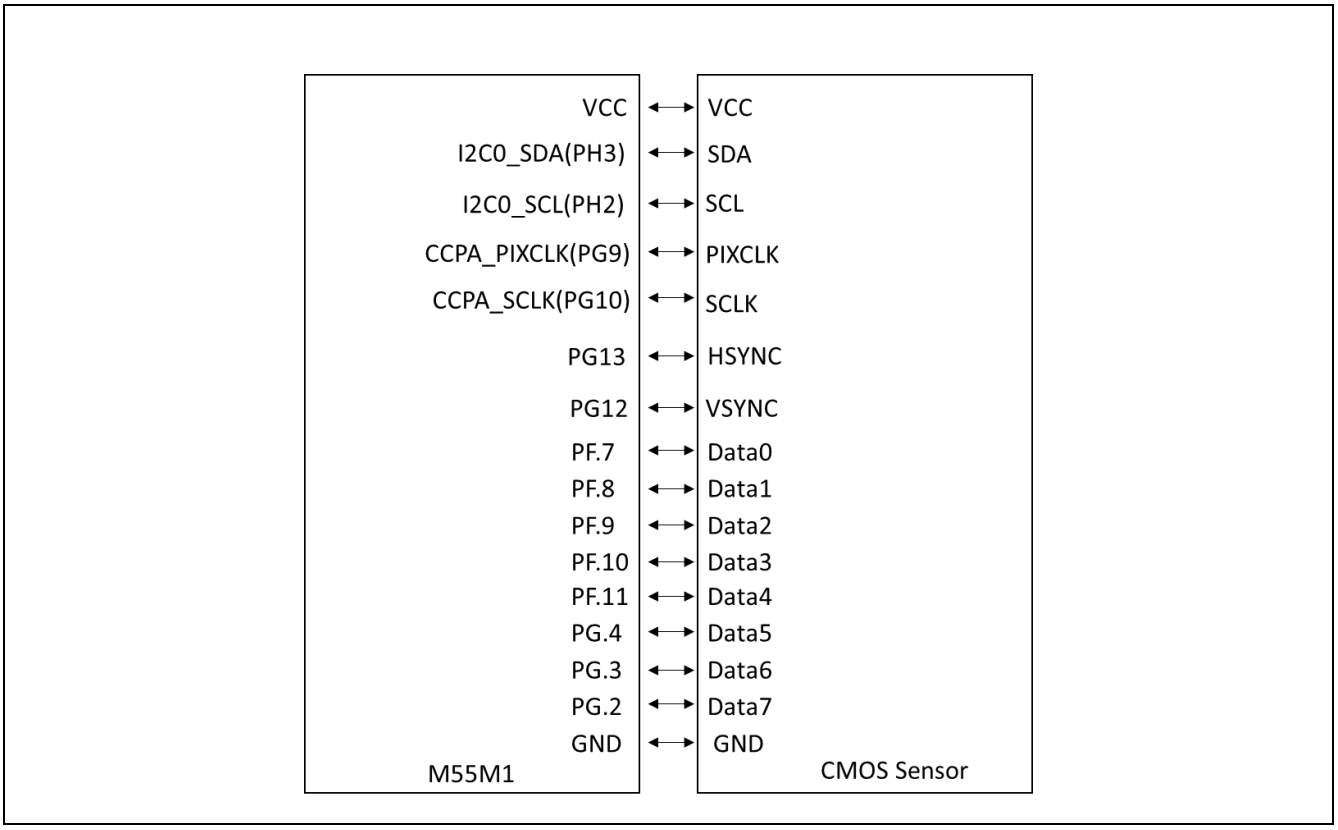

圖 3-2 CMOS 線路示意圖

4. 目錄資訊

目錄結構如下圖所示。

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

圖 4-1 目錄資訊

5. 範例程式執行

1. 根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，雙擊 *M55M1_Smart_Motion_Detection.uvprojx*。
2. 進入編譯模式介面
   - 編譯
   - 下載程式碼至記憶體
   - 進入 / 離開除錯模式
3. 進入除錯模式介面
   - 執行程式碼

6. 修訂紀錄

Date	Revision	Description
2024.06.24	1.00	初始發布。

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.