MA35D1 ECAP 模擬 PWM 捕捉模式

NuMicro® 64 位元系列微控制器範例代碼介紹

文件資訊

應用簡述	本範例代碼透過 ECAP 的功能來模擬 PWM Capture 模式
BSP 版本	Linux-6.6.y
開發平台	NuMaker-HMI-MA35D1 V1.0

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

在實際應用上，ECAP (Enhanced Input Capture Timer) 通常會搭配 QEI（Quadrature Encoder Interface）一起使用，特別常見於馬達相關的偵測與控制系統中。QEI 負責解碼編碼器 A/B 相訊號，取得馬達的位置與轉速資訊；藉由 ECAP 與 QEI 的結合，系統可以同時掌握馬達的機械位置與速度。

但是本範例程式實作了 ECAP 模組來模擬 PWM 捕捉模式功能，透過量測外部 PWM 或脈波訊號的頻率與占空比，進一步擴增 ECAP 在實際系統中的應用範圍與使用彈性。

MA35D1 內建的 ECAP 模組，可用來實現類似 EPWM Capture 模式的量測功能，用於量測外部 PWM 或脈波訊號的頻率與占空比等參數。

ECAP 模組提供多組獨立的 Input Capture，每一組具備：

- 一個可向上計數的 16 位元計時器/計數器
- 自動重載與比較匹配（Compare-Match）機制
- 三個獨立輸入捕捉通道

每個捕捉通道都可以偵測輸入腳位上的數位訊號邊緣變化（上升沿、下降沿或雙邊沿）。當偵測到設定的邊緣事件時，ECAP 會將當下的計數器值鎖存到對應的捕捉暫存器中。

藉由連續讀取不同邊緣（例如：上升沿到下一個上升沿或上升沿到下降沿）的捕捉計數值差，便可推算出：

- 輸入 PWM 週期（Period）
- 高電平時間（High Time）
- 低電平時間（Low Time）
- 占空比（Duty Cycle）

透過這種方式，ECAP 模組即可模擬 PWM 的輸入捕捉（Capture）模式，用於精確量測外部 PWM 或其他脈波訊號的時間特性，並可進一步配合中斷等機制，將量測結果自動搬移至記憶體，降低 CPU 負擔、提升量測效率。

1.1 原理

下圖紅線代表本範例量測 PWM 時，ECAP 所用的主要路徑，PWM 訊號從 ECAPn_IC0/1/2 腳位輸入，經過 Noise Filter 濾波後，透過 CAPSEL0/1/2 選擇對應到 CAP0、CAP1、CAP2，送入 Input Capture Control Unit。當偵測到設定的上升/下降沿時，ECAP 會把當下的 ECAP_CNT 計數值鎖存在 ECAP_HLD0/1/2 暫存器，用來後續計算 PWM 的週期與占空比。

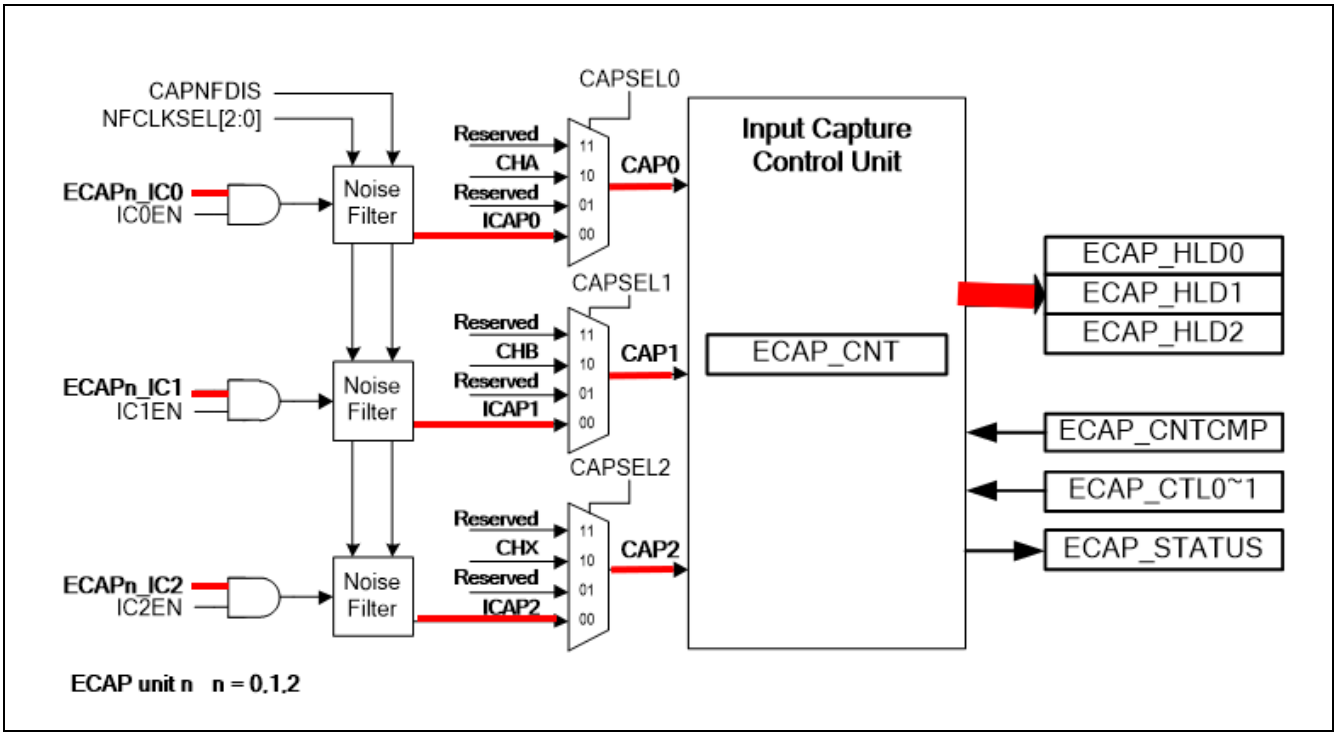

圖 1-1 ECAP 流程圖

ECAP（Input Capture Timer/Counter）透過「計時器 + 邊緣偵測」機制來量測 PWM 訊號在一個週期內的高電平時間與整體週期，進而計算占空比。

1. 訊號輸入與前級處理

- PWM 訊號由外部腳位輸入至 ECAP 的各通道（ECAPn_IC0、ECAPn_IC1、ECAPn_IC2）。
- 每個輸入通道先經過 Noise Filter（雜訊濾波），過濾毛邊與短暫的雜訊脈波，避免誤觸發邊緣事件。
- 經過濾波後的訊號再由多工器（CAPSEL0 / CAPSEL1 / CAPSEL2）選擇，對應到 CAP0、CAP1、CAP2 三組捕捉輸入路徑，送入 Input Capture Control Unit。

2. ECAP 計數器運作

- 內部共用一個 ECAP_CNT 計時器/計數器，以系統時鐘或經過預分頻後的時鐘作為計數來源。
- ECAP_CNT 通常設定為「向上計數模式」，從 0 開始往上加計數，計數值對應到「時間」的離散刻度。
- 當發生指定的輸入邊緣事件（上升沿或下降沿）時，目前的 ECAP_CNT 值會被截取到對應的暫存器（ECAP_HLDx）。

3. 邊緣設定與捕捉流程

為了量測 PWM 的占空比，需要取得「高電平時間」與「整個週期時間」：

- 週期（Period）：通常用「連續兩個相同邊緣」之間的時間差來量測，例如：由「上升沿 → 下一次上升沿」的 ECAP_CNT 差值。
- 高電平時間（High Time）：通常用「上升沿 → 下降沿」之間的時間差來量測。

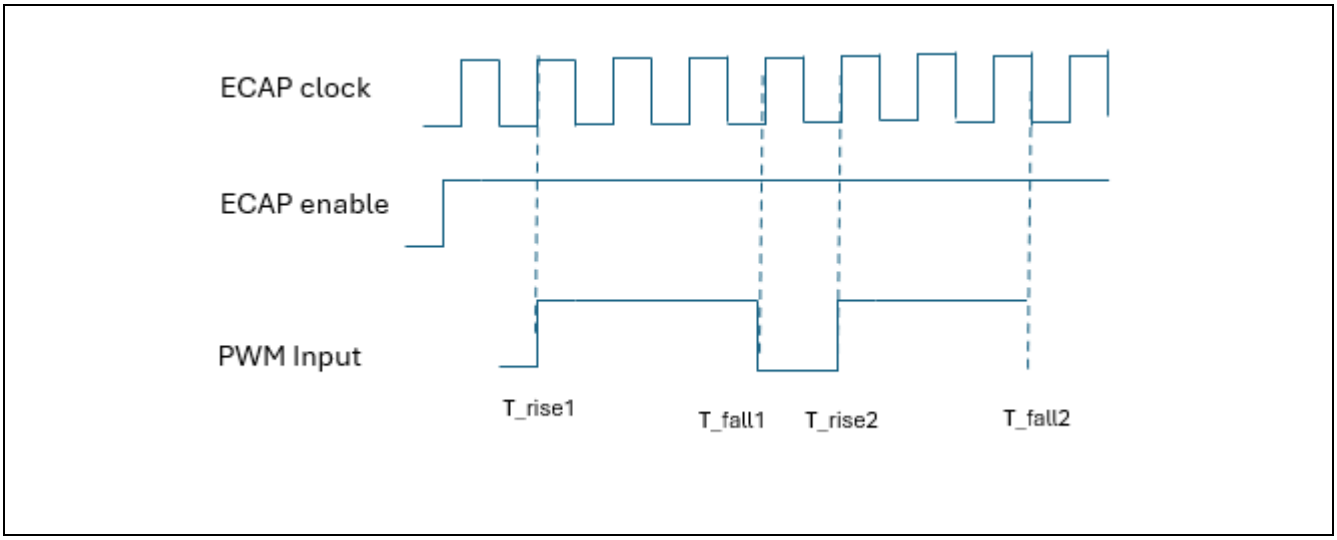

圖 1-2 時序圖

典型設定與流程如下（以 CAP0 為例）：

1) 邊緣觸發設定
    - 設定 CAP0 在「上升沿與下降沿（雙邊緣）」觸發捕捉。

- 每次捕捉事件發生時，同步取得輸入腳位的當下電位狀態，記錄到 ECAP_Level（用以判斷本次捕捉到的時間區間屬於 High 或 Low）。

2) 捕捉事件序列

- 選擇合適的捕捉事件序列（例如「上升 → 下降 → 上升」或「下降 → 上升 → 下降」等），由 Input Capture Control Unit 在事件發生時，將當下 ECAP_CNT 值鎖存至 ECAP_HLD0 暫存器。
- 每次捕捉事件鎖存 ECAP_CNT 到 ECAP_HLD0 後，硬體會將 ECAP_CNT 重置為 0，使下一段時間區間的量測從 0 重新開始，避免 16-bit 計數器因長時間計數而溢位（Overflow）。

3) 依時間順序取得兩段「區間時間」計數值（ticks）完成一次完整的捕捉序列後（以「上升 → 下降 → 上升」為例），可得到兩段相鄰邊緣之間的時間區間（ticks）：

- Δt1_ticks：第 1 段相鄰邊緣之間的計數值
  例如：上升沿 → 下降沿 的 ticks（鎖存在 ECAP_HLD0，CPU 讀出後存入 ECAP_TMP1）。
- Δt2_ticks：第 2 段相鄰邊緣之間的計數值
  例如：下降沿 → 下一次上升沿 的 ticks（鎖存在 ECAP_HLD0，CPU 讀出後存入 ECAP_TMP2）。

4) 計算週期與高/低電平時間

- 週期時間（Period，ticks）固定為兩段區間相加：
  Period_ticks = Δt1_ticks + Δt2_ticks
- 高電平時間（High Time，ticks）與低電平時間（Low Time，ticks）需依 ECAP_Level（或序列起始邊緣）判斷哪一段屬於 High、哪一段屬於 Low：

情況 A：第 1 段量到的是 High（例如序列為 上升→下降→上升）。

- High_ticks = Δt1_ticks
- Low_ticks = Δt2_ticks

情況 B：第 1 段量到的是 Low（例如序列為 下降→上升→下降，或起始電平不同）。

- High_ticks = Δt2_ticks

- Low_ticks = Δt1_ticks

5) 計算占空比（Duty Cycle） 占空比定義為「高電平時間 / 週期時間」，以百分比表示：
   - Duty(%) = (High_ticks / Period_ticks) × 100%
6) 換算成真實時間單位 只要已知 ECAP 計數器時脈 f_ECAP，即可將 ticks 換算為時間：
   - Time(s) = ticks / f_ECAP
   - 或 Time(us) = ticks × 10^6 / f_ECAP

4. 使用輪循的方式

以輪詢方式依序切換 IC0、IC1、IC2，逐一量測每一路輸入訊號的 period 與 duty。

5. 使用中斷或狀態暫存器輔助讀取

ECAP_STATUS / ECAP_CNTCMP / ECAP_CTL 等暫存器，用來表示捕捉事件發生、比較匹配狀態或控制模式。

當完成一次完整的捕捉序列（例如「上升 → 下降 → 上升」）後，可以觸發中斷或設置旗標，由 CPU 讀出 ECAP_HLDx 中的計數值並執行上述計算。為了比較有效率的方式，當驅動完成「上升 → 下降 → 上升」三次中斷，才通知使用者去讀取 period 與 duty。

1.2 執行結果

```
$ echo 1 > /sys/class/pwm/pwmchip0/export
$ echo 30000 > /sys/class/pwm/pwmchip0/pwm1/period
$ echo 20000 > /sys/class/pwm/pwmchip0/pwm1/duty_cycle
$ echo 1 > /sys/class/pwm/pwmchip0/pwm1/enable
$ cd /sys/class/pwm/pwmchip6/
$ echo 2 > export
# cat pwm2/capture
5398 3599
$
```

透過 PWM sysfs 設定 PWM0 輸出，並用另一個控制器（ECAP 模擬）做 capture 量測驗證。

1.2.1 PWM 輸出設定（pwmchip0/pwm1）

1. 進入目錄：*/sys/class/pwm/pwmchip0/* 。
2. 匯出 PWM0：echo 1 > export（建立 pwm1/ 節點）。
3. 設定參數：

- pwm1/period = 30000（通常單位 ns），週期 30 µs（約 33.33 kHz）。
- pwm1/duty_cycle = 20000（ns），高電平 20 µs，占空比約 66.7%。

4. 啟用輸出：echo 1 > pwm1/enable。

1.2.2 Capture 量測（pwmchip6/IC2）

1. 切換到 ECAP 模擬的 PWM 控制器：*/sys/class/pwm/pwmchip6/* 。
2. 本平台 export 與 IC 對應：
   - 1 → IC1
   - 2 → IC2
   - 3 → IC3
3. 匯出 IC2：echo 2 > export（對應 pwm2/）。
4. 讀取量測：cat pwm2/capture 。
   - 讀取結果為 5398 3599，可解讀為 capture 週期約 5398、High 約 3599（占空比約 66.7%）。

1.2.3 為何週期數值看起來不同（屬正常現象）

- PWM0 的 period/duty_cycle 是以「PWM 輸出端的時間單位（ns）」設定。
- pwmchip6:pwm2/capture 讀取值則以 capture 模組的取樣/計數時脈（sample clock / counter clock）為基準。

因為不同模組的時脈來源、除頻（prescaler）、解析度不同時，會造成「回讀週期數值」與 PWM 設定值在數字上不一致。本例中：3599/5398 ≈ 66.7% 與 20000/30000 ≈ 66.7% 一致，表示占空比關係正確。若需精確比對時間（ns/us），需依 capture 模組的 clock 與 prescaler 將讀取值換算後再比較。

2. 代碼介紹

2.1 主要 ECAP 驅動代碼

drivers/pwm/pwm-ma35d1-ecap.c

```
// SPDX-License-Identifier: GPL-2.0
/*
 * Driver for the Nuvoton MA35 series ECAP controller
 *
 * Copyright (C) 2025 Nuvoton Corporation
 *               Shan-Chun Hung <schung@nuvoton.com>
 */

#include <linux/clk.h>
#include <linux/io.h>
#include <linux/math64.h>
#include <linux/mod_devicetable.h>
#include <linux/module.h>
#include <linux/platform_device.h>
#include <linux/pwm.h>

#define TIMEOUT_CNT       1000000

#define ECAP_CTL0     0x14
#define ECAP_CTL1     0x18
#define ECAP_STATUS       0x1C

#define ECAP_HLD(ch)  (0x04+ch*4)

#define SET_EDGE_SEL(base, ch, edge) \
    writel((readl(base)&~0x3>>(ch<<1))|(edge<<(ch<<1)), base)

struct ma35_ecap {
    struct pwm_chip chip;
    void __iomem *base;
    struct clk *clk;
};

static inline struct ma35_ecap *to_ma35_ecap(struct pwm_chip *chip)
{
    return container_of(chip, struct ma35_ecap, chip);
}

static int ma35_ecap_capture(struct pwm_chip *chip, struct pwm_device *pwm,
                  struct pwm_capture *result, unsigned long timeout)
{
    struct ma35_ecap *ecap = to_ma35_ecap(chip);
    int ch = pwm->hwpwm;
    unsigned long timeout_cnt;
    int ret = 0;

    writel(0, ecap->base + ECAP_CTL0);
    writel(0, ecap->base + ECAP_CTL1);
```

```
49      writel(BIT(29) | BIT(4+ch), ecap->base + ECAP_CTL0);
50      writel(BIT(8+ch) | BIT(20+ch) | (BIT(1)<<(ch<<1)), ecap->base + ECAP_CTL1);
51
52      timeout_cnt = 0;
53      while ((readl(ecap->base + ECAP_STATUS) & BIT(8 + ch)))
54          if (++timeout_cnt > TIMEOUT_CNT)
55              goto TIMEOUT;
56
57      writel(BIT(ch), ecap->base + ECAP_STATUS);
58      writel(readl(ecap->base + ECAP_CTL0) | BIT(24), ecap->base + ECAP_CTL0);
59      timeout_cnt = 0;
60      while ((readl(ecap->base + ECAP_STATUS) & BIT(ch)) != BIT(ch))
61          if (++timeout_cnt > TIMEOUT_CNT)
62              goto TIMEOUT;
63
64      writel(BIT(ch), ecap->base + ECAP_STATUS);
65      timeout_cnt = 0;
66      while ((readl(ecap->base + ECAP_STATUS) & BIT(ch)) != BIT(ch))
67          if (++timeout_cnt > TIMEOUT_CNT)
68              goto TIMEOUT;
69
70      writel(BIT(ch), ecap->base + ECAP_STATUS);
71      result->duty_cycle = readl(ecap->base + ECAP_HLD(ch));
72
73      timeout_cnt = 0;
74      while ((readl(ecap->base + ECAP_STATUS) & BIT(ch)) != BIT(ch))
75          if (++timeout_cnt > TIMEOUT_CNT)
76              goto TIMEOUT;
77
78      result->period = result->duty_cycle + readl(ecap->base + ECAP_HLD(ch));
79      return ret;
80
81  TIMEOUT:
82      result->period = 0;
83      result->duty_cycle = 0;
84      return 0;
85  }
86
87  static int ma35_pwm_apply(struct pwm_chip *chip, struct pwm_device *pwm,
88                const struct pwm_state *state)
89  {
90      return 0;
91  }
92
93  static const struct pwm_ops ma35_ecap_ops = {
94      .apply = ma35_pwm_apply,
95      .capture = ma35_ecap_capture,
96      .owner = THIS_MODULE,
97  };
98
99  static int ma35_ecap_probe(struct platform_device *pdev)
100 {
101     struct ma35_ecap *ecap;
102     int ret;
103
104     ecap = devm_kzalloc(&pdev->dev, sizeof(*ecap), GFP_KERNEL);
```

```
105       if (IS_ERR(ecap))
106           return PTR_ERR(ecap);
107
108       ecap->base = devm_platform_ioremap_resource(pdev, 0);
109       if (IS_ERR(ecap->base))
110           return PTR_ERR(ecap->base);
111
112       ecap->clk = of_clk_get(pdev->dev.of_node, 0);
113       if (IS_ERR(ecap->clk)) {
114           dev_err(&pdev->dev, "failed to get epwm clock\n");
115           ret = PTR_ERR(ecap->clk);
116           return -ENOENT;
117       }
118
119       ret = clk_prepare_enable(ecap->clk);
120       if (ret)
121           return -ENOENT;
122
123       platform_set_drvdata(pdev, ecap);
124       ecap->chip.dev = &pdev->dev;
125       ecap->chip.ops = &ma35_ecap_ops;
126       ecap->chip.base = -1;
127       ecap->chip.npwm = 3;
128
129       ret = pwmchip_add(&ecap->chip);
130       if (ret < 0)
131           return dev_err_probe(&pdev->dev, ret, "unable to add ecap chip");
132
133       return 0;
134   }
135
136   static const struct of_device_id ma35_ecap_of_match[] = {
137       {.compatible = "nuvoton,ma35d1-ecap" },
138       { }
139   };
140
141   MODULE_DEVICE_TABLE(of, ma35_ecap_of_match);
142
143   static struct platform_driver ma35_ecap_driver = {
144       .probe = ma35_ecap_probe,
145       .driver = {
146               .name = "ma35-ecap",
147               .of_match_table = ma35_ecap_of_match,
148                },
149   };
150
151   module_platform_driver(ma35_ecap_driver);
152
153   MODULE_LICENSE("GPL");
154   MODULE_AUTHOR("Shan-Chun Hung <schung@nuvoton.com>");
155   MODULE_ALIAS("platform:ma35-ecap");
```

- 這支檔案實作的是 Nuvoton MA35 系列 ECAP 控制器的 Linux driver。
- 它把 ECAP 模組包裝成一個 pwm_chip，掛到 Linux PWM framework 底下。

- 主要功能是：
  - 只做「PWM capture」量測（ma35_ecap_capture()），
  - 沒有實作產生 PWM 波形（ma35_pwm_apply() 只是回傳 0）。

所以實務上，它讓上層（其他驅動或 user space 工具）可以呼叫 PWM 子系統，來量測外部 PWM 訊號的「週期 + 占空比」。

1. 把 ECAP 變成一顆「PWM 量測器」，也就是將 ECAP 驅動和 Linux PWM 子系統註冊

- 在 ma35_ecap_probe() 裡：
  - 取得 ECAP 的暫存器位址（ecap->base）和 clock（ecap->clk）。
  - 填好 ecap->chip，宣告有 3 個 channel。
  - 用 pwmchip_add() 把它註冊到 Linux PWM framework。
- 之後系統就把這顆 ECAP 當成一個 PWM 控制器，上層可以呼叫 .capture 來「量測 PWM」，而不是產生 PWM。

2. ma35_ecap_capture() 怎麼量測 PWM

一次 capture 的流程大致是：

- 清乾淨設定與舊狀態
  - 把 ECAP_CTL0、ECAP_CTL1 清 0，再重新設定對應 channel 的 capture 模式與邊緣。
  - 把 ECAP_STATUS 相關 bit 清掉，避免上一輪的事件干擾。
- 啟動 ECAP 計數器，等 3 次邊緣事件
  - 開啟計數器（ECAP_CTL0 的 enable bit）。
  - 依序等 3 次 ECAP_STATUS 的事件 bit 變成 1，每次都代表「又捕捉到一個邊緣」：
    - 第 1 次：只用來同步，程式不讀值。
    - 第 2 次：讀 ECAP_HLD(ch)，當成 result->duty_cycle（高電平時間）。

  - 第 3 次：再讀一次 ECAP_HLD(ch)，和前面的 duty_cycle 相加，得到 result->period（完整週期）。

- 逾時就回傳 0
  - 如果在任何等待迴圈裡超過 TIMEOUT_CNT，就把 period 和 duty_cycle 都設成 0，表示這次沒量到有效 PWM。

2.2 在 PWM 子系統中加入 MA35D1 ECAP 驅動

drivers/pwm/Makefile

```
obj-$(CONFIG_PWM_MA35D1_ECAP)        += pwm-ma35d1-ecap.o
```

這一行的作用：

- 當 Kconfig 選到 CONFIG_PWM_MA35D1_ECAP=y 或 m 時，
- 會把 *pwm-ma35d1-ecap.o* 編進核心或編成模組，
- 也就是把你寫的 ma35 ECAP driver 納入 PWM framework 的編譯流程。

drivers/pwm/Kconfig

```
config PWM_MA35D1_ECAP
        tristate "Nuvoton MA35D1 ECAP support"
        help
          Captures wareform driver for Nuvoton MA35D1.

          To compile this driver as a module, choose M here: the module
          will be called pwm-ma35d1-ecap.
```

- 定義一個新的 Kconfig 選項 PWM_MA35D1_ECAP。
- tristate 代表可以選 y/m/n：
  - y：內建進核心
  - m：編成模組 *pwm-ma35d1-ecap.ko*
  - n：不編譯
- help 文字說明這個 driver 是用來做 MA35D1 的波形 capture。

2.3 在 Device Tree 中宣告 ECAP 硬體節點

arch/arm64/boot/dts/nuvoton/ma35d1.dtsi

```
ecap0: ecap0@40b40000 {
        compatible = "nuvoton,ma35d1-ecap";
        reg = <0x0 0x40b40000 0x0 0x10000>;
        clocks = <&clk ECAP0_GATE>;
        status = "disabled";
};
```

這段在 .dtsi 裡面定義 ECAP0 這個硬體區塊：

- ecap0@40b40000：ECAP0 的 MMIO 基底位址是 0x40b40000，size 0x10000。
- compatible = "nuvoton,ma35d1-ecap";
  對應到你 driver 裡的 ma35_ecap_of_match，讓 kernel 能用這個字串比對到正確的驅動。
- clocks = <&clk ECAP0_GATE>;
  指定使用哪一條 clock gate。
- status = "disabled";
  預設先關閉，實際板子用的時候再在 .dts 裡把它打開。

2.4 在 SOM 板的 DTS 中設定腳位與啟用 ECAP0

ma35d1-som-256m.dts 的 pinctrl 設定

在 &pinctrl { } 底下新增：

```
ecap0 {
        pinctrl_ecap0: ecap0grp{
                nuvoton,pins =
                        <SYS_GPD_MFPH_PD13MFP_ECAP0_IC0 &pcfg_default>,
                        <SYS_GPD_MFPH_PD14MFP_ECAP0_IC1 &pcfg_default>,
                        <SYS_GPD_MFPH_PD15MFP_ECAP0_IC2 &pcfg_default>;
        };
};
```

這一段做的事是：

- 建立一個 pinctrl group pinctrl_ecap0。
- 把 PD13 / PD14 / PD15 這三個實體腳位切換成 ECAP0 的三個輸入功能：
  - PD13 → ECAP0_IC0

- PD14 → ECAP0_IC1
- PD15 → ECAP0_IC2

- &pcfg_default 表示這些腳使用預設的電氣屬性（上拉/下拉，驅動能力等）。

也就是說，這裡把外部的 PWM/脈波訊號「接線」到 ECAP0 三個 capture 通道。

在同一個 DTS 裡新增 ECAP0 節點

```
&ecap0 {
        status = "okay";
        #pwm-cells = <2>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_ecap0>;
};
```

這一段把前面在 .dtsi 宣告的 ecap0 實際打開：

- status = "okay";
  告訴 kernel：這顆 ECAP0 在這塊板子上是有接出來要用的。
- #pwm-cells = <2>;
  告訴 PWM framework 這個控制器的 PWM spec 需要 2 個 cell（例如 <channel period> 這種格式），和常見 PWM bindings 一致。
- pinctrl-names = "default";
- pinctrl-0 = <&pinctrl_ecap0>;
  啟動時套用剛才設定好的 pin group，讓腳位真正切換到 ECAP 功能。

3. 軟體與硬體需求

3.1 軟體需求

- BSP 版本
  - Linux-6.6.y
- IDE 版本
  - Buildroot_2024

3.2 硬體需求

- 電路元件
  - NuMaker-HMI-MA35D1 V1.0

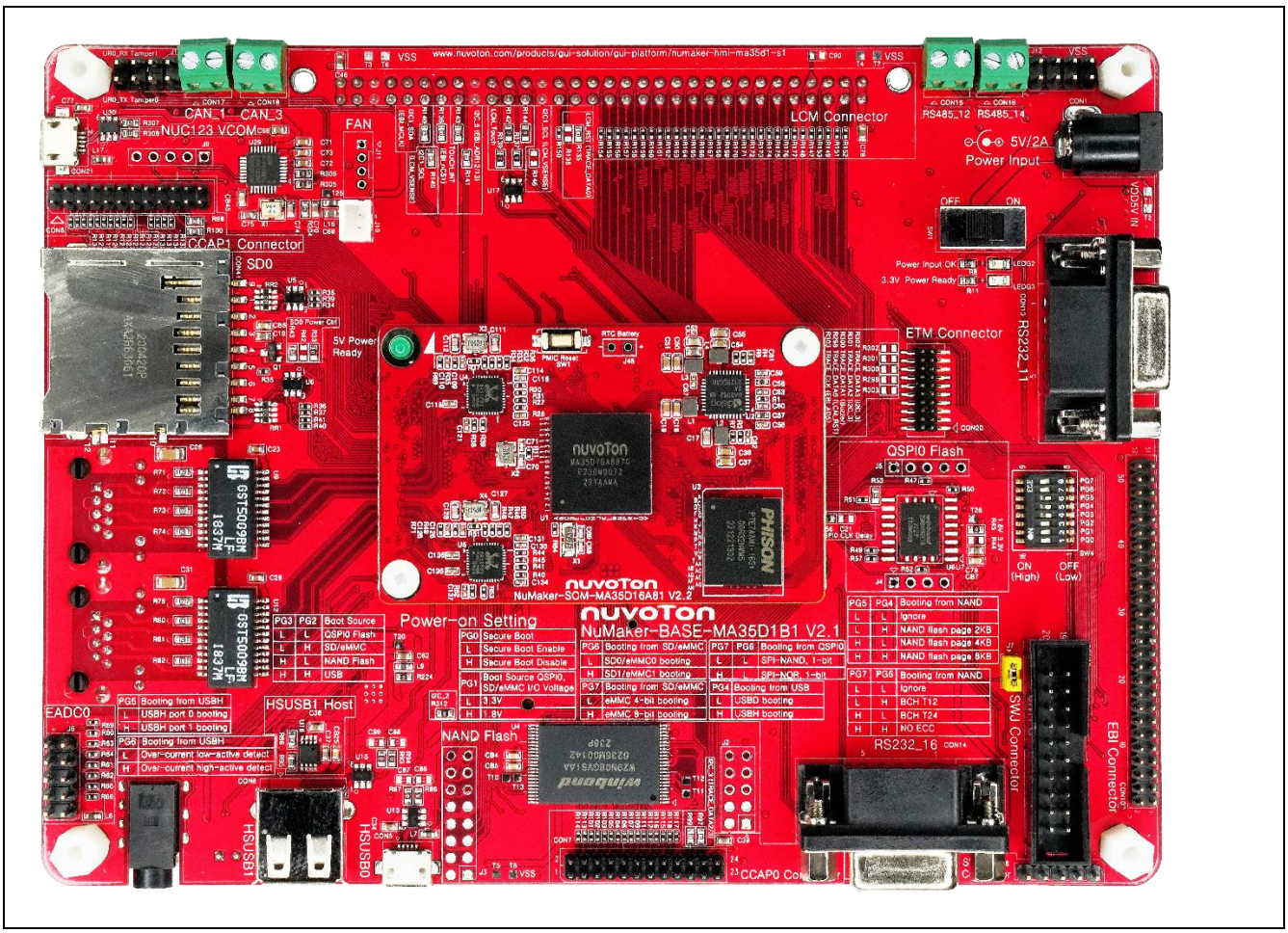

圖 3-1 NuMaker-HMI-MA35D1 V1.0

- 線路示意圖
  - 藍色框框為 ECAP0_IC1(PD14)
  - 黃色框框為 ECAP0_IC0(PD15)

圖 3-2 ECAP Pins 位置

- LCD connector 第 37 腳目前是 EPWM CH1，可透過外部連線接到 ECAP0_IC0（PD15），即可將 EPWM 輸出的 PWM 訊號輸入到 ECAP0_IC0 進行量測；其他腳位也可依此方式對應連接到 ECAP 其餘輸入腳。由於這是一種較特殊的接線與應用方式，目前並沒有對應的標準板子可直接使用，需要自行配線實作。

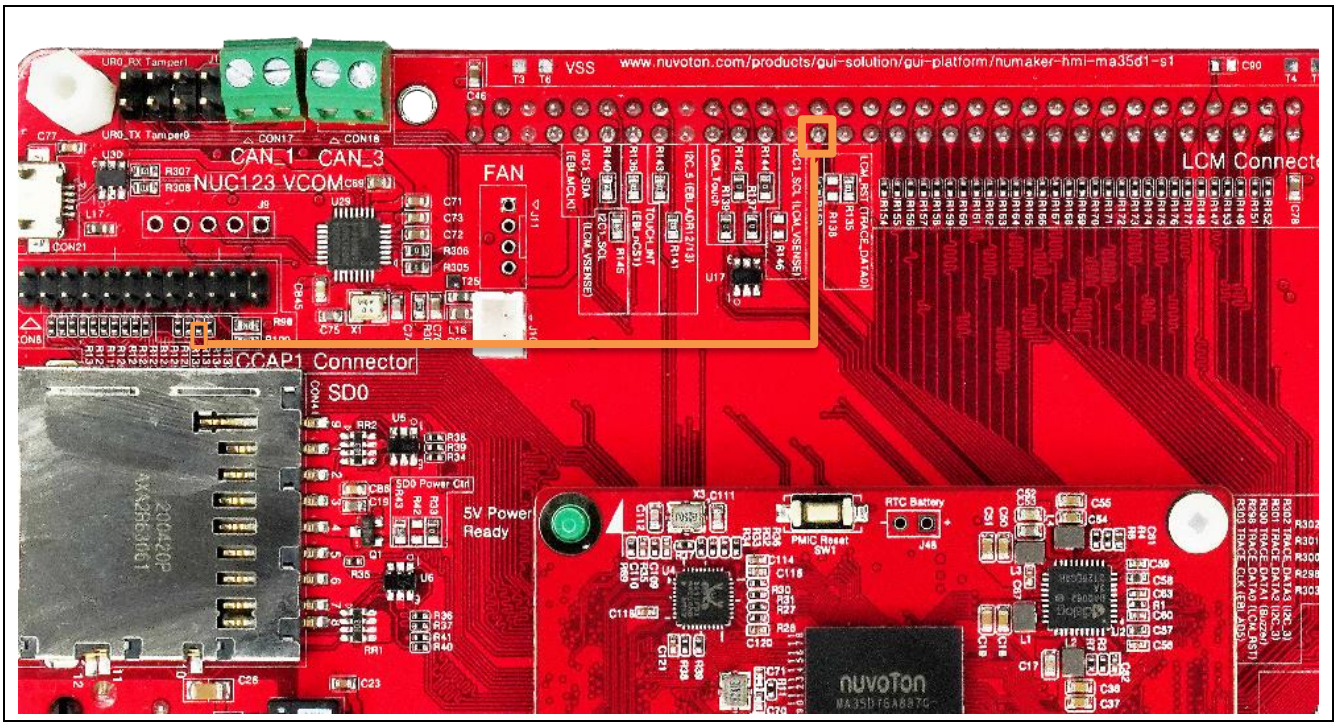

圖 3-3 EPWM_CH1 接 ECAP0_IC0

4. 目錄資訊

目錄結構如下圖所示。

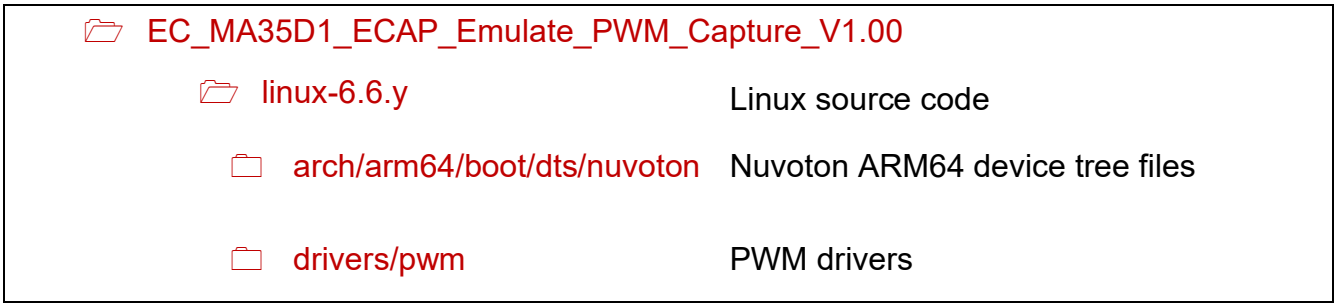

圖 4-1 目錄資訊

5. 範例程式執行

MA35D1 預設會編譯 Linux 映像，範例要能成功執行必須修改 linux 資料夾底下的相關 source code，以下提供 buildroot 相關步驟，外部存儲裝置為 SD card:

1. 進入 MA35D1 buildroot 後先做第一次的編譯，才會產生相關程式源碼與映像檔。

```
user@bb867ccf5905:~/shared/buildroot_2024$ make numaker_som_ma35d16a81_defconfig
user@bb867ccf5905:~/shared/buildroot_2024$ make
```

2. 進入 linux-6.6.y 相關路徑(根據個人設定會有所不同):

```
user@bb867ccf5905:~/shared/buildroot_2024/output/build/linux-custom$
```

3. 將範例程式中各別 linux-6.6.y 資料夾內的所有檔案複製，並取代原 linux-custom 底下的對應檔案。**注意：**該動作是取代，並不是直接將範例程式資料夾直接整個複製到linux-custom 資料夾。

4. 進入 MA35D1 linux-6.6.y 相關選單。

```
user@bb867ccf5905:~/shared/buildroot_2024$ make linux-menuconfig
```

5. 在 linux-menuconfig 中從 PWM 裡面開啟 ECAP 功能。

```
Device Drivers  --->
[*] Pulse-Width Modulation (PWM) Support  --->
<*>   Nuvoton MA35D1 ECAP support
```

6. 重新編譯 linux-custom。

```
make linux-rebuild
```

7. 重新編譯映像檔。

```
make
```

8. 透過 NuWriter 燒錄 pack 映像。

9. 將開發板上的 Power-on setting 指撥開關設定為 boot from SD0，啟動 MA35D1 開發板。

6. 修訂紀錄

Date	Revision	Description
2026.04.13	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.