MA35D1 ECAP 仿真 PWM 捕捉模式

NuMicro® 64 位系列微控制器范例代码介绍

文件信息

应用简述	本范例代码通过 ECAP 的功能来仿真 PWM Capture 模式
BSP 版本	Linux-6.6.y
开发平台	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

在实际应用上，ECAP (Enhanced Input Capture Timer) 通常会搭配 QEI（Quadrature Encoder Interface）一起使用，特别常见于马达相关的侦测与控制系统中。QEI 负责译码编码器 A/B 相信号，取得马达的位置与转速信息；藉由 ECAP 与 QEI 的结合，系统可以同时掌握马达的机械位置与速度。

但是本范例程序实作了 ECAP 模块来仿真 PWM 捕捉模式功能，通过测量外部 PWM 或脉冲信号的频率与占空比，进一步扩增 ECAP 在实际系统中的应用范围与使用弹性。

MA35D1 内建的 ECAP 模块，可用来实现类似 EPWM Capture 模式的测量功能，用于测量外部 PWM 或脉冲信号的频率与占空比等参数。

ECAP 模块提供多组独立的 Input Capture，每一组具备：

- 一个可向上计数的 16 位定时器/计数器
- 自动重载与比较匹配（Compare-Match）机制
- 三个独立输入捕捉通道

每个捕捉通道都可以侦测输入脚位上的数字信号边缘变化（上升沿、下降沿或双边沿）。当侦测到设定的边缘事件时，ECAP 会将当下的计数器值锁存到对应的捕捉缓存器中。

藉由连续读取不同边缘（例如：上升沿到下一个上升沿或上升沿到下降沿）的捕捉计数值差，便可推算出：

- 输入 PWM 周期（Period）
- 高电平时间（High Time）
- 低电平时间（Low Time）
- 占空比（Duty Cycle）

通过这种方式，ECAP 模块即可仿真 PWM 的输入捕捉（Capture）模式，用于精确测量外部 PWM 或其他脉冲信号的时间特性，并可进一步配合中断等机制，将测量结果自动搬移至内存，降低 CPU 负担、提升测量效率。

1.1 原理

下图红线代表本范例测量 PWM 时，ECAP 所用的主要路径，PWM 信号从 ECAPn_IC0/1/2 脚位输入，经过 Noise Filter 滤波后，通过 CAPSEL0/1/2 选择对应到 CAP0、CAP1、CAP2，送入 Input Capture Control Unit。当侦测到设定的上升/下降沿时，ECAP 会把当下的 ECAP_CNT 计数值锁存在 ECAP_HLD0/1/2 寄存器，用来后续计算 PWM 的周期与占空比。

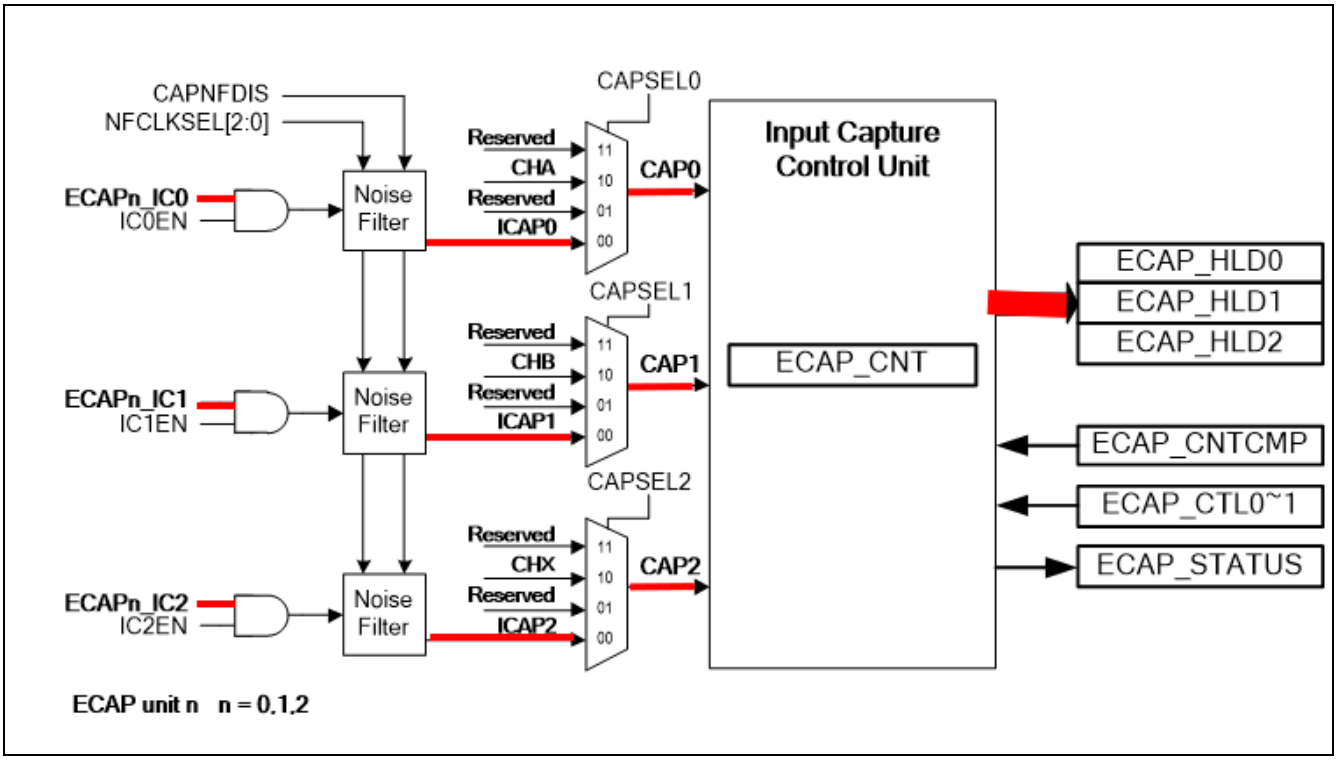

图 1-1 ECAP 流程图

ECAP（Input Capture Timer/Counter）通过「定时器 + 边缘侦测」机制来测量 PWM 信号在一个周期内的高电平时间与整体周期，进而计算占空比。

1. 信号输入与前级处理

   - PWM 信号由外部脚位输入至 ECAP 的各通道（ECAPn_IC0、ECAPn_IC1、ECAPn_IC2）。
   - 每个输入信道先经过 Noise Filter（噪声滤波），过滤毛边与短暂的噪声脉波，避免误触发边缘事件。
   - 经过滤波后的信号再由多任务器（CAPSEL0 / CAPSEL1 / CAPSEL2）选择，对应到 CAP0、CAP1、CAP2 三组捕捉输入路径，送入 Input Capture Control Unit。

2. ECAP 计数器运作

- 内部共享一个 ECAP_CNT 定时器/计数器，以系统时钟或经过预分频后的时钟作为计数来源。
- ECAP_CNT 通常设定为「向上计数模式」，从 0 开始往上加计数，计数值对应到「时间」的离散刻度。
- 当发生指定的输入边缘事件（上升沿或下降沿）时，目前的 ECAP_CNT 值会被截取到对应的缓存器（ECAP_HLDx）。

3. 边缘设定与捕捉流程

为了测量 PWM 的占空比，需要取得「高电平时间」与「整个周期时间」：

- 周期（Period）：通常用「连续两个相同边缘」之间的时间差来测量，例如：由「上升沿 → 下一次上升沿」的 ECAP_CNT 差值。
- 高电平时间（High Time）：通常用「上升沿 → 下降沿」之间的时间差来测量。

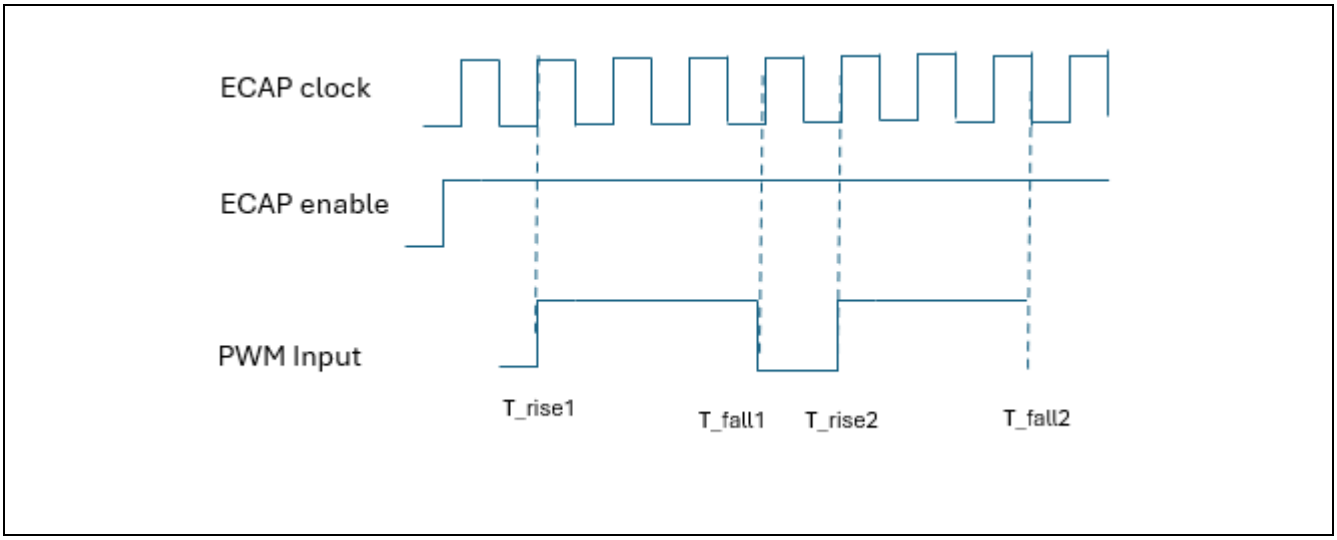

图 1-2 时序图

典型设定与流程如下（以 CAP0 为例）：

1) 边缘触发设定

- 设定 CAP0 在「上升沿与下降沿（双边缘）」触发捕捉。

- 每次捕捉事件发生时，同步取得输入脚位的当下电位状态，记录到 ECAP_Level（用以判断本次捕捉到的时间区间属于 High 或 Low）。

2) 捕捉事件序列

- 选择合适的捕捉事件序列（例如「上升 → 下降 → 上升」或「下降 → 上升 → 下降」等），由 Input Capture Control Unit 在事件发生时，将当下 ECAP_CNT 值锁存至 ECAP_HLD0 寄存器。
- 每次捕捉事件锁存 ECAP_CNT 到 ECAP_HLD0 后，硬件会将 ECAP_CNT 重置为 0，使下一段时间区间的测量从 0 重新开始，避免 16-bit 计数器因长时间计数而溢位（Overflow）。

3) 依时间顺序取得两段「区间时间」计数值（ticks）完成一次完整的捕捉序列后（以「上升 → 下降 → 上升」为例），可得到两段相邻边缘之间的时间区间（ticks）：

- Δt1_ticks：第 1 段相邻边缘之间的计数值
  例如：上升沿 → 下降沿 的 ticks（锁存在 ECAP_HLD0，CPU 读出后存入 ECAP_TMP1）。
- Δt2_ticks：第 2 段相邻边缘之间的计数值
  例如：下降沿 → 下一次上升沿 的 ticks（锁存在 ECAP_HLD0，CPU 读出后存入 ECAP_TMP2）。

4) 计算周期与高/低电平时间

- 周期时间（Period，ticks）固定为两段区间相加：
  Period_ticks = Δt1_ticks + Δt2_ticks
- 高电平时间（High Time，ticks）与低电平时间（Low Time，ticks）需依 ECAP_Level（或序列起始边缘）判断哪一段属于 High、哪一段属于 Low：

  情况 A：第 1 段量到的是 High（例如序列为 上升→下降→上升）。
  - High_ticks = Δt1_ticks
  - Low_ticks = Δt2_ticks

  情况 B：第 1 段量到的是 Low（例如序列为 下降→上升→下降，或起始电平不同）。
  - High_ticks = Δt2_ticks
  - Low_ticks = Δt1_ticks

5) 计算占空比（Duty Cycle） 占空比定义为「高电平时间 / 周期时间」，以百分比表示：

- Duty(%) = (High_ticks / Period_ticks) × 100%

6) 换算成真实时间单位 只要已知 ECAP 计数器频率 f_ECAP，即可将 ticks 换算为时间：

- Time(s) = ticks / f_ECAP
- 或 Time(us) = ticks × 10^6 / f_ECAP

4. 使用轮循的方式

以轮询方式依序切换 IC0、IC1、IC2，逐一测量每一路输入信号的 period 与 duty。

5. 使用中断或状态缓存器辅助读取

ECAP_STATUS / ECAP_CNTCMP / ECAP_CTL 等缓存器，用来表示捕捉事件发生、比较匹配状态或控制模式。

当完成一次完整的捕捉序列（例如「上升 → 下降 → 上升」）后，可以触发中断或设置旗标，由 CPU 读出 ECAP_HLDx 中的计数值并执行上述计算。为了比较有效率的方式，当驱动完成「上升 → 下降 → 上升」三次中断，才通知使用者去读取 period 与 duty。

1.2 执行结果

```
$ 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
$
```

通过 PWM sysfs 设定 PWM0 输出，并用另一个控制器（ECAP 模拟）做 capture 测量验证。

1.2.1 PWM 输出设定（pwmchip0/pwm1）

1. 进入目录：*/sys/class/pwm/pwmchip0/* 。
2. 汇出 PWM0：echo 1 > export（建立 pwm1/ 节点）。
3. 设定参数：

- pwm1/period = 30000（通常单位 ns），周期 30 µs（约 33.33 kHz）。
- pwm1/duty_cycle = 20000（ns），高电平 20 µs，占空比约 66.7%。

4. 启用输出：echo 1 > pwm1/enable。

1.2.2 Capture 测量（pwmchip6/IC2）

1. 切换到 ECAP 模拟的 PWM 控制器：*/sys/class/pwm/pwmchip6/* 。
2. 本平台 export 与 IC 对应：
   - 1 → IC1
   - 2 → IC2
   - 3 → IC3
3. 汇出 IC2：echo 2 > export（对应 pwm2/）。
4. 读取测量：cat pwm2/capture 。
   - 读取结果为 5398 3599，可解读为 capture 周期约 5398、High 约 3599（占空比约 66.7%）。

1.2.3 为何周期数值看起来不同（属正常现象）

- PWM0 的 period/duty_cycle 是以「PWM 输出端的时间单位（ns）」设定。
- pwmchip6:pwm2/capture 读取值则以 capture 模块的取样/计数频率（sample clock / counter clock）为基准。

因为不同模块的频率来源、除频（prescaler）、分辨率不同时，会造成「回读周期数值」与 PWM 设定值在数字上不一致。本例中：3599/5398 ≈ 66.7% 与 20000/30000 ≈ 66.7% 一致，表示占空比关系正确。若需精确比对时间（ns/us），需依 capture 模块的 clock 与 prescaler 将读取值换算后再比较。

2. 代码介绍

2.1 主要 ECAP 驱动代码

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");
```

- 该档案实现的是 Nuvoton MA35 系列 ECAP 控制器的 Linux driver。
- 它把 ECAP 模块包装成一个 pwm_chip，挂到 Linux PWM framework 底下。

- 主要功能是：
  - 只做「PWM capture」测量（ma35_ecap_capture()），
  - 没有实作产生 PWM 波形（ma35_pwm_apply() 只是回传 0）。

所以实务上，它让上层（其他驱动或 user space 工具）可以呼叫 PWM 子系统，来测量外部 PWM 信号的「周期 + 占空比」。

1. 把 ECAP 变成一颗「PWM 测量器」，也就是将 ECAP 驱动和 Linux PWM 子系统注册

- 在 ma35_ecap_probe() 里：
  - 取得 ECAP 的缓存器地址（ecap->base）和 clock（ecap->clk）。
  - 填好 ecap->chip，定义有 3 个 channel。
  - 用 pwmchip_add() 把它注册到 Linux PWM framework。
- 之后系统就把这颗 ECAP 当成一个 PWM 控制器，
  上层可以呼叫 .capture 来「测量 PWM」，而不是产生 PWM。

2. ma35_ecap_capture() 怎么测量 PWM

一次 capture 的流程大致是：

- 清干净设定与旧状态
  - 把 ECAP_CTL0、ECAP_CTL1 清 0，再重新设定对应 channel 的 capture 模式与边缘。
  - 把 ECAP_STATUS 相关 bit 清掉，避免上一轮的事件干扰。
- 启动 ECAP 计数器，等 3 次边缘事件
  - 开启计数器（ECAP_CTL0 的 enable bit）。
  - 依序等 3 次 ECAP_STATUS 的事件 bit 变成 1，每次都代表「又捕捉到一个边缘」：
    - 第 1 次：只用来同步，程序不读值。
    - 第 2 次：读 ECAP_HLD(ch)，当成 result->duty_cycle（高电平时间）。
    - 第 3 次：再读一次 ECAP_HLD(ch)，和前面的 duty_cycle 相加，得到 result->period（完整周期）。

- 超时就回传 0
  - 如果在任何等待循环里超过 TIMEOUT_CNT，就把 period 和 duty_cycle 都设成 0，表示这次没量到有效 PWM。

2.2 在 PWM 子系统中加入 MA35D1 ECAP 驱动

drivers/pwm/Makefile

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

这一行的作用：

- 当 Kconfig 选到 CONFIG_PWM_MA35D1_ECAP=y 或 m 时，
- 会把 *pwm-ma35d1-ecap.o* 编进核心或编成模块，
- 也就是把你写的 ma35 ECAP driver 纳入 PWM framework 的编译流程。

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.
```

- 定义一个新的 Kconfig 选项 PWM_MA35D1_ECAP。
- tristate 代表可以选 y/m/n：
  - y：内建进核心
  - m：编成模块 *pwm-ma35d1-ecap.ko*
  - n：不编译
- help 文字说明这个 driver 是用来做 MA35D1 的波形 capture。

2.3 在 Device Tree 中定义 ECAP 硬件节点

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";
```

```
};
```

这段在 .dtsi 里面定义 ECAP0 这个硬件区块：

- ecap0@40b40000：ECAP0 的 MMIO 基地址是 0x40b40000，size 0x10000。
- compatible = "nuvoton,ma35d1-ecap";
  对应到你 driver 里的 ma35_ecap_of_match，让 kernel 能用这个字符串比对到正确的驱动。
- clocks = <&clk ECAP0_GATE>;
  指定使用哪一条 clock gate。
- status = "disabled";
  预设先关闭，实际板子用的时候再在 .dts 里把它打开。

2.4 在 SOM 板的 DTS 中设定脚位与启用 ECAP0

ma35d1-som-256m.dts 的 pinctrl 设定

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>;
        };
};
```

这一段做的事是：

- 建立一个 pinctrl group pinctrl_ecap0。
- 把 PD13 / PD14 / PD15 这三个实体脚位切换成 ECAP0 的三个输入功能：
  - PD13 → ECAP0_IC0
  - PD14 → ECAP0_IC1
  - PD15 → ECAP0_IC2
- &pcfg_default 表示这些脚使用默认的电气属性（上拉/下拉，驅動能力等）。

也就是说，这里把外部的 PWM/脉冲信号「接线」到 ECAP0 三个 capture 通道。

在同一个 DTS 里新增 ECAP0 节点

```
&ecap0 {
        status = "okay";
        #pwm-cells = <2>;
```

```
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_ecap0>;
};
```

这一段把前面在 .dtsi 定义的 ecap0 实际打开：

- status = "okay";
  告诉 kernel：这颗 ECAP0 在这块板子上是有接出来要用的。
- #pwm-cells = <2>;
  告诉 PWM framework 这个控制器的 PWM spec 需要 2 个 cell（例如 <channel period> 这种格式），和常见 PWM bindings 一致。
- pinctrl-names = "default";
- pinctrl-0 = <&pinctrl_ecap0>;
  启动时套用刚才设定好的 pin group，让脚位真正切换到 ECAP 功能。

3. 软件与硬件需求

3.1 软件需求

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

3.2 硬件需求

- 电路组件
  - NuMaker-HMI-MA35D1 V1.0

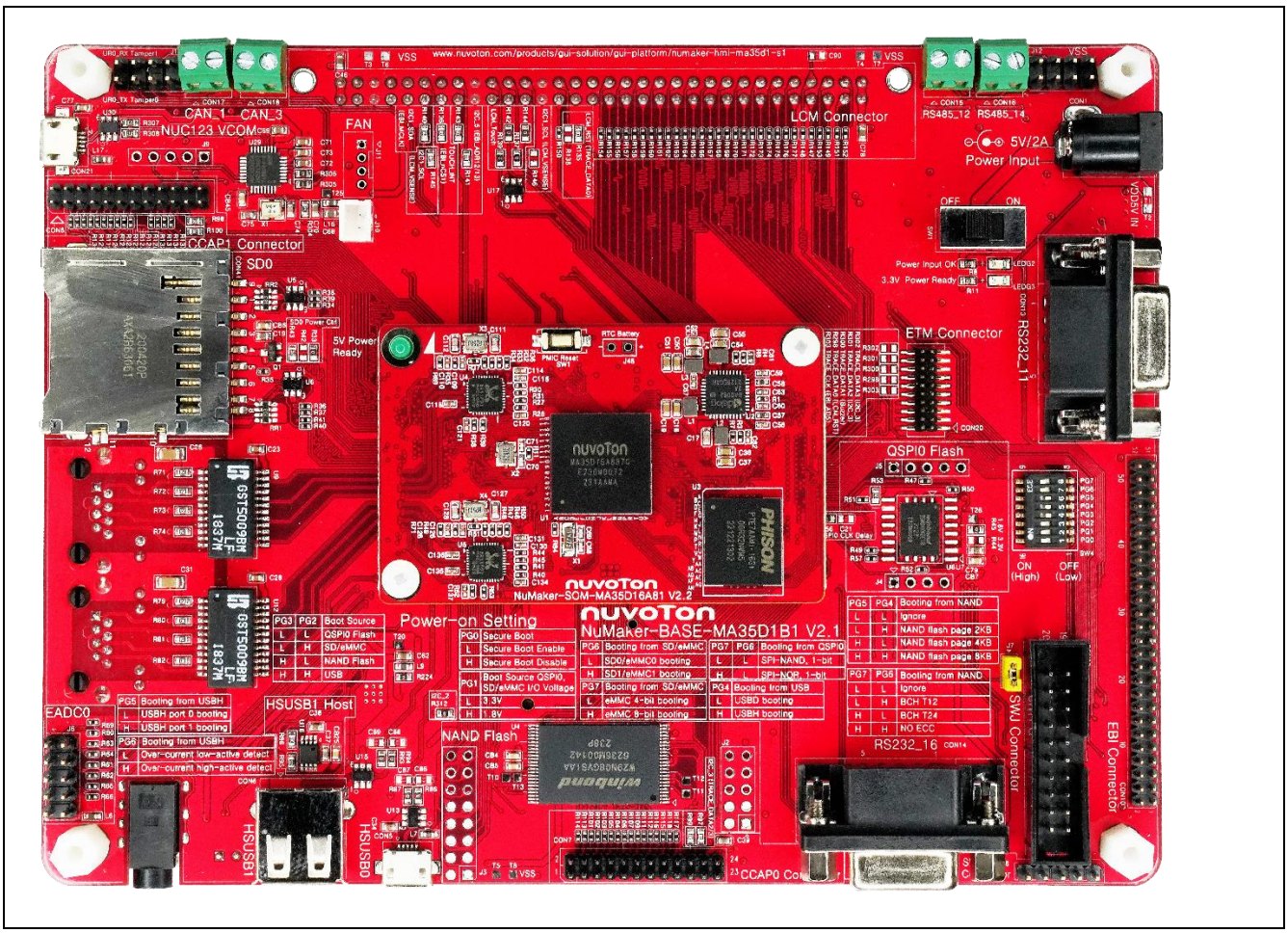

图 3-1 NuMaker-HMI-MA35D1 V1.0

- 线路示意图
  - 蓝色框框为 ECAP0_IC1(PD14)
  - 黄色框框为 ECAP0_IC0(PD15)

图 3-2 ECAP Pins 位置

- LCD connector 第 37 脚目前是 EPWM CH1，可通过外部联机接到 ECAP0_IC0（PD15），即可将 EPWM 输出的 PWM 信号输入到 ECAP0_IC0 进行测量；其他脚位也可依此方式对应连接到 ECAP 其余输入脚。由于这是一种较特殊的接线与应用方式，目前并没有对应的标准板子可直接使用，需要自行配线实作。

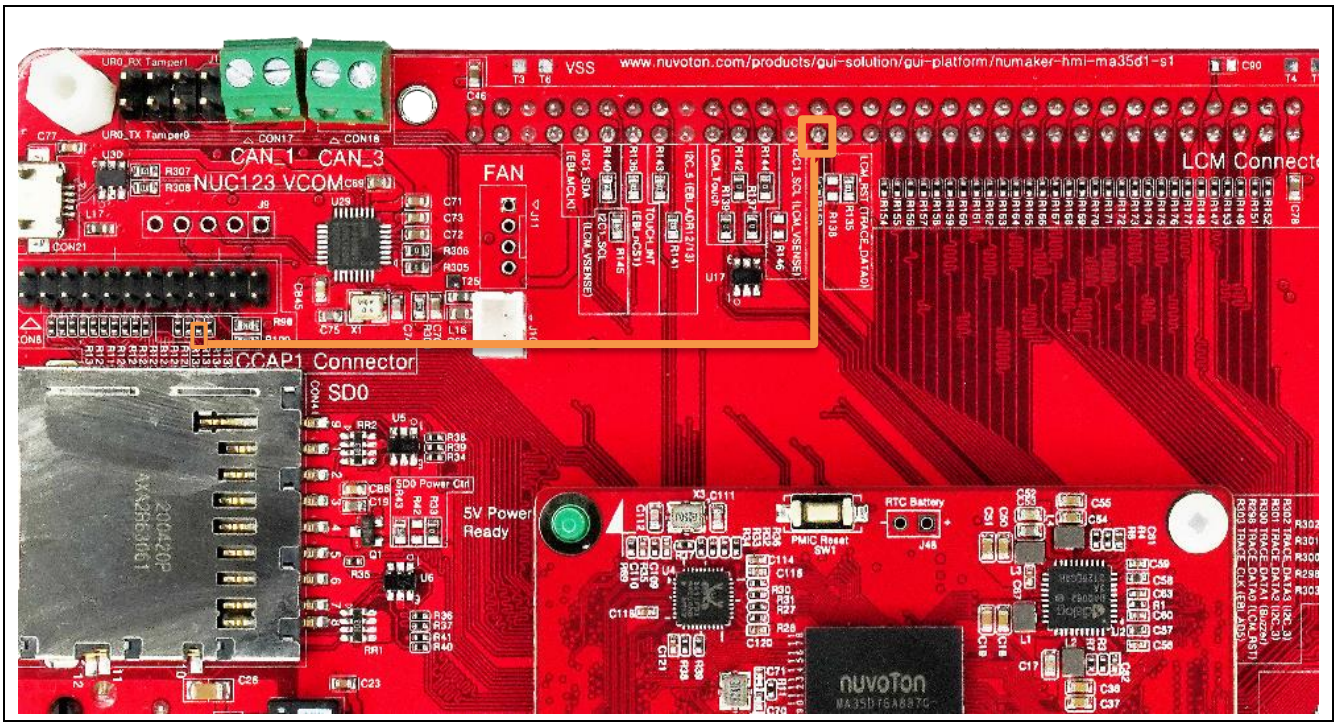

图 3-3 EPWM_CH1 接 ECAP0_IC0

4. 目录信息

目录结构如下图所示。

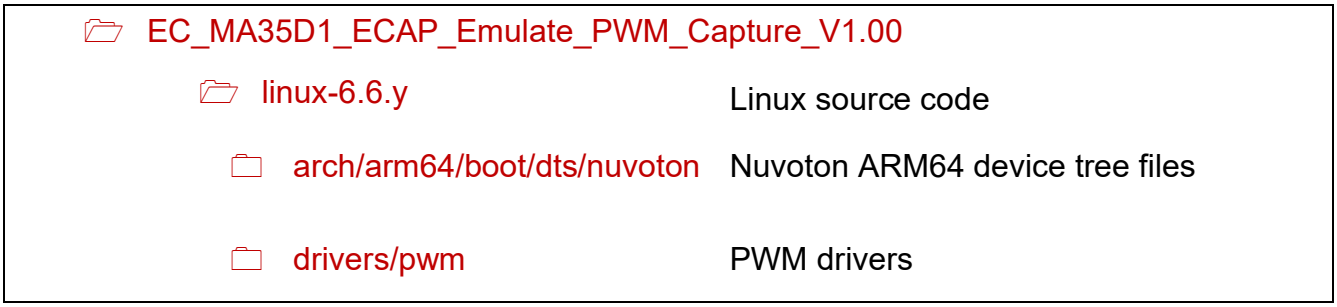

图 4-1 目录信息

5. 范例程序执行

MA35D1 预设会编译 Linux 映像，范例要能成功执行必须修改 linux 文件夹底下的相关 source code，以下提供 buildroot 相关步骤，外部存储装置为 SD card:

1. 进入 MA35D1 buildroot 后先做第一次的编译，才会产生相关程序源码与映像档。

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

2. 进入 linux-6.6.y 相关路径(根据个人设定会有所不同):

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

3. 将范例程序中各别 linux-6.6.y 文件夹内的所有档案复制，并取代原 linux-custom 底下的对应档案。**注意**：该动作是取代，并不是直接将范例程序文件夹直接整个复制到linux-custom 文件夹。

4. 进入 MA35D1 linux-6.6.y 相关选单。

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

5. 在 linux-menuconfig 中从 PWM 里面开启 ECAP 功能。

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

6. 重新编译 linux-custom。

```
make linux-rebuild
```

7. 重新编译映像档。

```
make
```

8. 通过 NuWriter 刻录 pack 映像。

9. 将开发板上的 Power-on setting 指拨开关设定为 boot from SD0，启动 MA35D1 开发板。

6. 修订纪录

Date	Revision	Description
2026.04.13	1.00	初始发布。

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.