F28P55x编程实例Labs-EQEP

- Code Composer Studio
- C2000Ware
- LaunchXL-F28P55x

EQEP

GPIO	PIN 脚	用途
EQEP_A	J12-EQEP1A	PWM模拟EQEPA
EQEP_B	J12-EQEP1B	PWM模拟EQEPB
EQEP_Index	J12-EQEP1I	GPIO模拟Index

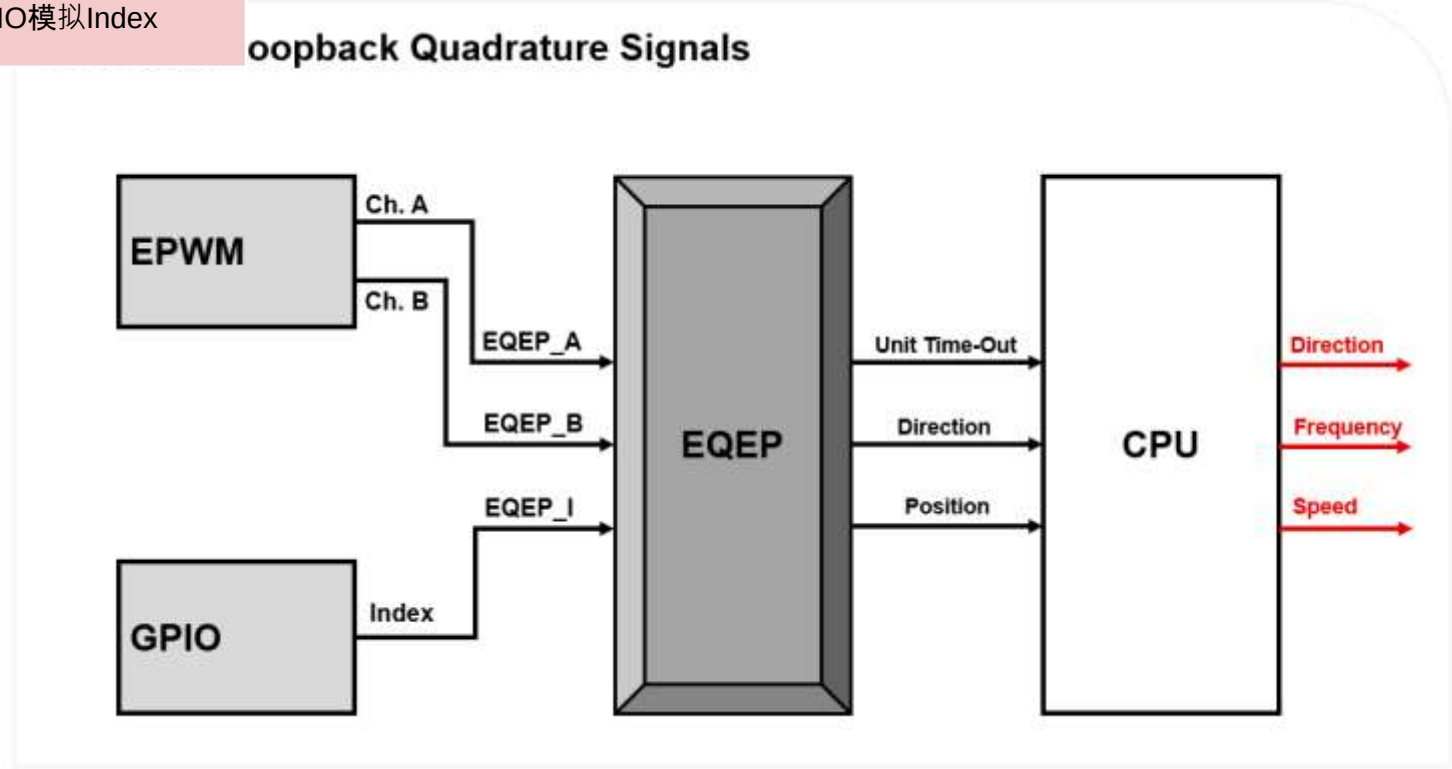
功能实现

用片上资源EPWM产生方波A和方波B,
用GPIO口模拟Index，以此来模拟编码器

用EQEP实现方向、频率等信息的采集

实现步骤

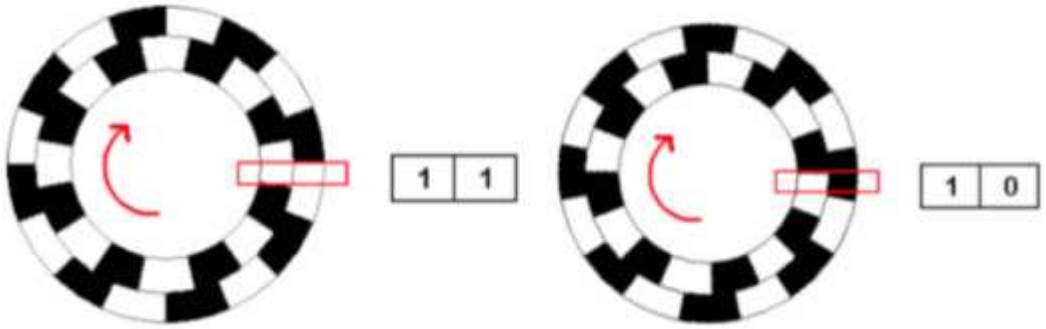
- 复制空白工程
- Sysconfig配置EPWM的A/B通道
- Sysconfig配置GPIO输入
- Sysconfig配置EQEP资源

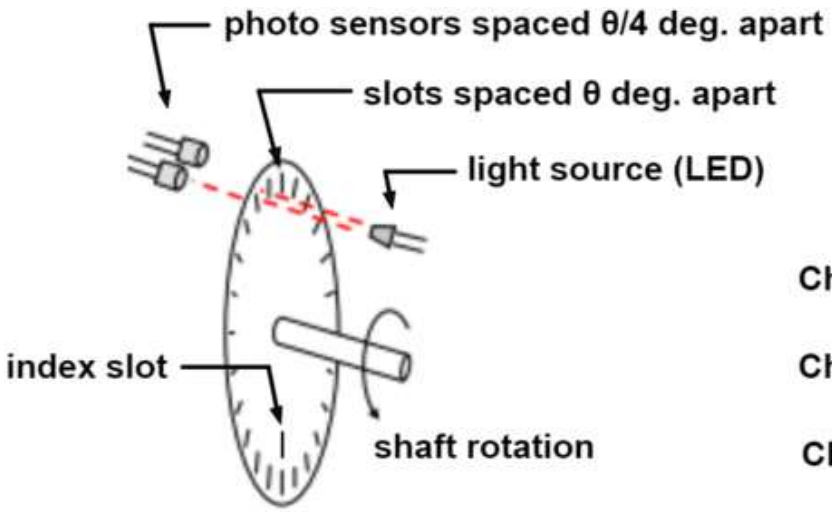


EQEP

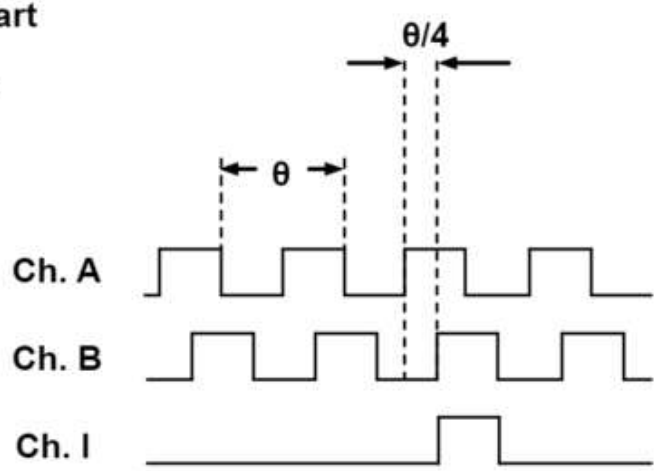
Enhanced Quadrature Encoder Pulse, 增强型正交编码脉冲

- 位置
- 方向
- 速度

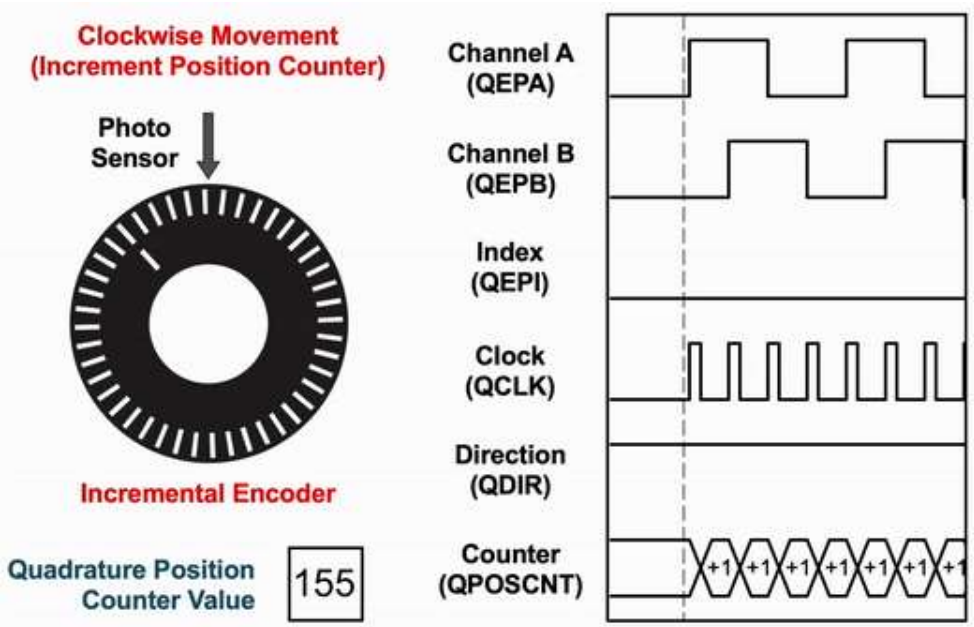




Incremental Optical Encoder



Quadrature Output from Photo Sensors



EQEP

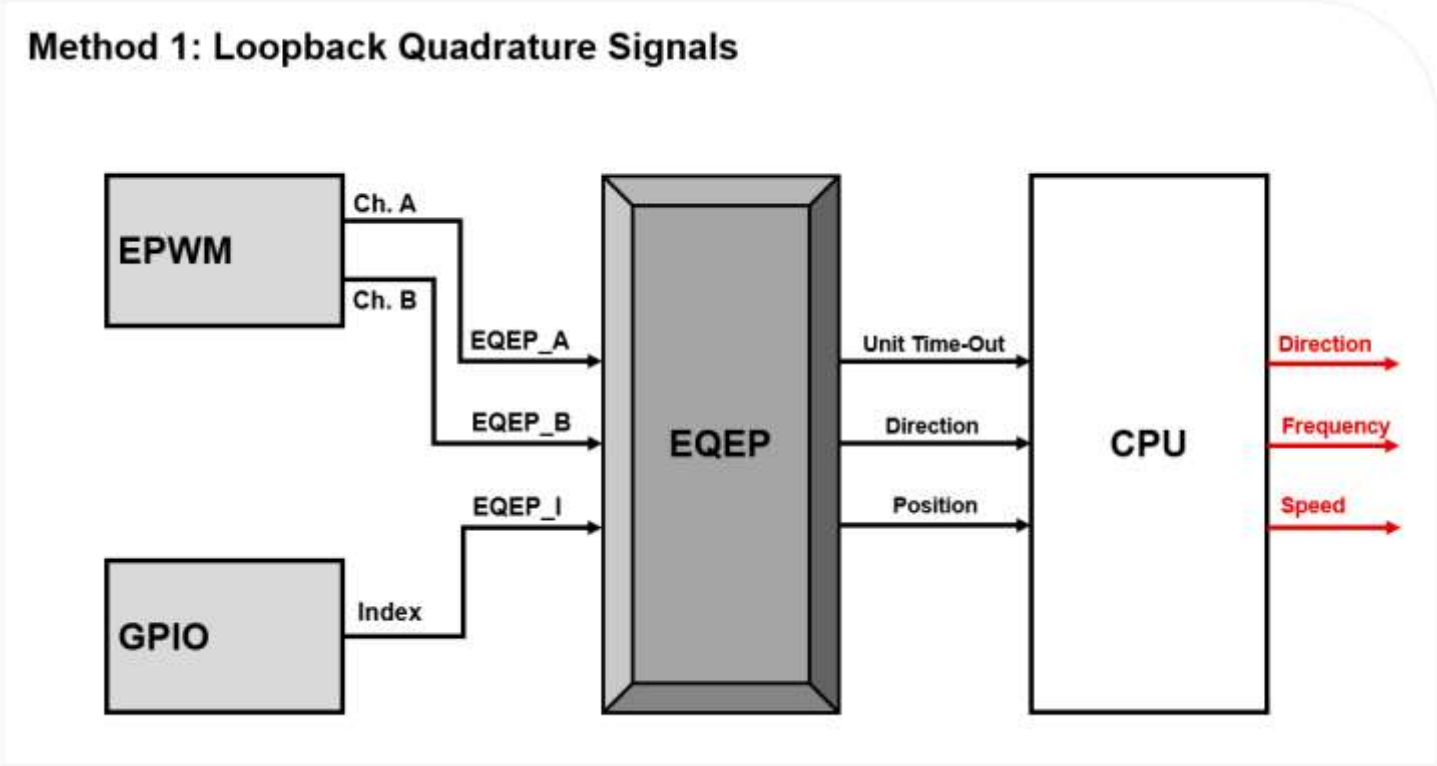
功能实现

用片上资源EPWM产生方波A和方波B，
用GPIO口模拟Index，以此来模拟编码器

用EQEP实现方向、频率等信息的采集

实现步骤

- 复制空白工程
- Sysconfig配置EPWM的A/B通道
- Sysconfig配置GPIO输入
- Sysconfig配置EQEP资源



EQEP

```
//
// Included Files
//
#include "driverlib.h"
#include "device.h"
#include "board.h"
```

```
//
// Defines
//
#define ENCODER_SLOTS   1000U          // LVSERVOMTR is a 1000-line encoder
#define UNIT_PERIOD     10000U         // Unit period in microseconds
```

```
//
// Globals
//
uint32_t oldCount = 0;                 // stores the previous position counter value
uint32_t newCount = 0;                 // stores the new position counter value for frequency calculation
uint32_t currentEncoderPosition = 0;    // stores the current encoder position
int32_t frequency = 0;                 // measured quadrature signal frequency of motor using eQEP
float32_t speed = 0.0f;                // measured speed of motor in rpm
int32_t direction = 0;                // direction of rotation of motor
```

```
//
// Main
//
void main(void)
{
                                //
    // Initialize device clock and peripherals
    //
    Device_init();
    //
    // Disable pin locks and enable internal pull ups.
    //
    Device_initGPIO();
    //
    // Initialize PIE and clear PIE registers. Disables CPU interrupts.
    //
    Interrupt_initModule();
    //
    // Initialize the PIE vector table with pointers to the shell Interrupt
    // Service Routines (ISR).
    //
    Interrupt_initVectorTable();
    // Board Initialization
                        Board_init();
    //
    // Enable Global Interrupt (INTM) and realtime interrupt (DBGM)
    //
    EINT;
    ERTM;
    //
    // Loop indefinitely
    //
    while(1)
    {
        //
        // myGPIOIndex pulses high for 200 microseconds every 1000 encoder cycles (400,000 us)
        //
        DEVICE_DELAY_US(400000L);
        GPIO_writePin(myGPIOIndex, 1);
        DEVICE_DELAY_US(200L);
        GPIO_writePin(myGPIOIndex, 0);
    }
}
```

EQEP

```
INT_myEQEP1_ISR(void)
{
    //
    // Save current encoder position value for monitoring
    //
    currentEncoderPosition = EQEP_getPosition(myEQEP1_BASE);
    //
    // Get position counter value latched on unit time-out event
    //
    newCount = EQEP_getPositionLatch(myEQEP1_BASE);
    //
    // Gets rotation direction of motor
    //
    direction = EQEP_getDirection(myEQEP1_BASE);
    //
    // Calculates the number of position in unit time based on direction
    //
    if (direction > 0 ){
        if (newCount >= oldCount)
            newCount = newCount - oldCount;
        else
            newCount = ((4 * ENCODER_SLOTS - 1) - oldCount) + newCount;
        }
    else {
        if (newCount <= oldCount)
            newCount = oldCount - newCount;
        else
            newCount = ((4 * ENCODER_SLOTS - 1) - newCount) + oldCount;
        }
    //
    // Stores the current position count value to oldCount variable
    //
    oldCount = currentEncoderPosition;
    //
    // Calculate frequency and speed values
    // frequency found by counting external input pulses for UNIT PERIOD
    // speed derived from encoder frequency and encoder resolution
    //
    frequency = (newCount * (uint32_t)1000000U) / ((uint32_t)UNIT_PERIOD);
    speed = (frequency * 60) / ((float)(4 * ENCODER_SLOTS));
    //
    // Clear interrupt flag and issue ACK
    //
    EQEP_clearInterruptStatus(myEQEP1_BASE,EQEP_INT_UNIT_TIME_OUT|EQEP_INT_GLOBAL);
    Interrupt_clearACKGroup(INT_myEQEP1_INTERRUPT_ACK_GROUP);
}
```