НАВЧАЛЬНО-МЕТОДИЧНА ЛІТЕРАТУРА

Тернопільський національний технічний університет імені Івана Пулюя

Кафедра автоматизації технологічних процесів і виробництв

Лабораторна робота № R03

з курсу

”Мікропроцесорні та програмні засоби автоматизації”

Розширені функції роботи з портами Raspberry Pi

Тернопіль 2021

Методичні вказівки до лабораторної роботи № R03 “ Розширені функції роботи з портами Raspberry Pi” з курсу "Мікропроцесорні та програмні засоби автоматизації". Медвідь В.Р., Пісьціо В.П.,, Тернопіль: ТНТУ, 20211 - 10 с.

Для студентів напряму: 151 "Автоматизація та комп'ютерні технології"

Автори: Медвідь В.Р., Пісьціо В.П.

Тема роботи

Розширені функції роботи з портами Raspberry Pi

Мета роботи

Ознайомитись з розширеними функціональними можливостями та роботою із GPIO Raspberry Pi за допомогою бібліотеки WiringPi.

Теоретичні відомості

Спеціфічні функції вводу-виводу Raspberry Pi

Дані функції не є частиною основного набору wiringPi , але діють спеціально на апаратну частину Raspberry Pi. Деякі модулі зовнішніх апаратних драйверів можуть надавати частину цієї функціональності.

- void digitalWriteByte(int value); підпрограма записує байт, що виводиться на перші 8 контактів GPIO. Це найшвидший спосіб встановити всі 8 біт одразу, хоча все-таки потрібно дві операції запису.
- pwmSetMode(int mode); ШІМ-генератор може працювати в двох режимах - «збалансований» (коректний за фазою) та «починати з нуля». Другий режим є традиційним, проте режим за замовчуванням у Pi "коректний за фазою". Ви можете перемикати режими, подаючи параметр:PWM_MODE_BAL або PWM_MODE_MS .
- pwmSetRange (unsigned int range); встановлює діапазон змінних у в ШІМ. За замовчуванням діапазон рівний 1024. (вхідна змінна 0-1023)
- pwmSetClock (int divisor); встановлює дільник для тактового сигналу ШІМ. Функції керування ШІМ не можна використовувати в режимі Sys.
- piBoardRev (void); повертає версію плати Raspberry Pi. Деякі контакти BCM_GPIO змінили номер і функцію при переході на іншу версію плати, тому якщо використовується номери контактів BCM_GPIO, потрібно знати про відмінності.
- wpiPinToGpio (int wPiPin); повертає номер контакту BCM_GPIO даного контакту wiringPi.
- physPinToGpio (int physPin); повертає номер контакту BCM_GPIO для даного фізичного контакту на роз'ємі P1 .
- setPadDrive (група int, значення int); встановлює "потужність" драйверів ліній для певної групи ліній. Є 3 групи контактів, а потужність - від 0 до 7. Не використовуйте цю функцію, якщо ви не знаєте, що робите.
- void pwmWrite (int pin, int значення); записує значення в регістр ШІМ для даного контакту. Raspberry Pi має одну вбудовану лінію PWM, контакт 1 (BMC_GPIO 18, Phys 12), і діапазон - 0-1024. Інші пристрої ШІМ можуть мати інші діапазони ШІМ. Ця функція не може керувати лініює ШІМ Pi у режимі Sys .

Програмний ШІМ

WiringPi включає в себе програмний обробник ШІМ, здатний виводити сигнал ШІМ на будь-який з GPIO-контактів Raspberry Pi. Існують деякі обмеження… Для підтримки низького використання процесора мінімальна ширина імпульсу становить 100 мкс. У поєднанні із запропонованим за замовчуванням діапазоном 100 отримується частота ШІМ 100 Гц. Ви можете зменшити діапазон, щоб отримати більш високу частоту, за рахунок роздільної здатності, або збільшити останню, але це знизить частоту. Якщо ви зміните ширину імпульсу в коді драйвера, то майте на увазі, що при затримках менше 100 мкс wiringPi робить це в програмному циклі, а це означає, що використання процесора різко зросте, а керувати більш ніж одним контактом буде майже неможливим.

Також зауважте, що хоча root виконуються з більш високим пріоритетом у режимі реального часу, Linux все ще може впливати на точність генерованого сигналу. Однак у цих обмеженнях управління світлом / світлодіодом або двигуном є досяжним.

Використання:

```
#include <wiringPi.h>
#include <softPwm.h>
```

При компіляції програми необхідно включити PTHREAD бібліотеку, а також

wiringPibібліотеки:

```
cc -o myprog myprog.c -lwiringPi -lpthread
```

Ви повинні ініціалізувати wiringPi за допомогою однієї з функцій wiringPiSetup () , wiringPiSetupGpio () або wiringPiSetupPhys () . wiringPiSetupSys () недостатньо швидкий, тому потрібно запускати свої програми з sudo.

Деякі модулі розширення також можуть бути досить швидкими для обробки програмного забезпечення ШІМ - це було протестовано, наприклад, з розширенням GPP MCP23S17 на PiFace.

Доступні наступні дві функції:

int softPwmCreate (int pin, int InitialValue, int pwmRange); створює програмно керований контакт ШІМ. Можна використовувати будь-який контакт GPIO, і нумерація контактів буде такою, яку використовувала функція wiringPiSetup (). Використовуйте 100 для pwmRange , тоді значення може бути від 0 (вимкнено) до 100 (повністю включено) для даного контакту. Повернене значення - 0 для успіху. Все інше, і вам слід перевірити глобальну змінну errno, щоб побачити, що пішло не так.

void softPwmWrite (int pin, int значення); оновлює значення ШІМ на даному контакті. Перевіряється значення в межах діапазону, і контакти, які раніше не були ініціалізовані через softPwmCreate, будуть ігноруватися.

Кожен "цикл" виходу ШІМ займає 10 мс із значенням діапазону за замовчуванням 100, тому намагання змінити значення ШІМ більше 100 разів на секунду буде марним.

Кожен контакт, активований у режимі softPWM, використовує приблизно 0,5% ЦП. Зараз немає можливості відключити softPWM на контакті під час роботи програми. Вам потрібно виконувати програму, щоб підтримувати вихід ШІМ

Часові функції

В той час як Linux надає безліч системних викликів та функцій для надання різноманітних функцій синхронізації та режиму сну, іноді це може бути дуже заплутано, особливо якщо ви новачок у Linux, тому представлені тут функції імітують ті, що доступні на платформі Arduino, роблячи перенесення та написання новий код дещо простішим. Навіть якщо ви не використовуєте жодну з функцій вводу / виводу, вам все одно потрібно викликати одну з функцій налаштування wiringPi - просто використовуйте wiringPiSetupSys (), якщо вам не потрібен кореневий доступ у вашій програмі, і пам'ятайте, що #include <wiringPi .h>

unsigned int millis (void) повертає число, що представляє кількість мілісекунд, з часу першого звертання до одної з функцій wiringPiSetup. Вона повертає 32-бітове число без знаку, яке повторюється через 49 днів.

unsigned int micros (void) що представляє кількість мікросекунд, з часу першого звертання до одної з функцій wiringPiSetup.. Віна повертає 32-бітове число без знаку, яке повторюється через 71 хвилину.

void delay (unsigned int howLong) призупиняє виконання програми принаймні як на час howLong мілісекунд. Внаслідок багатозадачності Linux тривалість інтервалу може бути довше. Зауважте, що максимальна затримка - це 32-бітове ціле число без знаку або приблизно 49 днів.

void delayMicroseconds (unsigned int howLong) зупиняє виконання програми як мінімум на howLong мікросекунд .Внаслідок багатозадачності Linux це може бути довше. Зауважте, що максимальна затримка - це 32-бітне число мікросекунд або приблизно 71 хвилина. Затримки менше 100 мікросекунд призначаються за допомогою жорстко закодованого циклу, який постійно запитує системний час; Затримки понад 100 мікросекунд виконуються за допомогою функції nanosleep () системи. Можливо, вам доведеться врахувати наслідки дуже коротких затримок на загальну продуктивність системи, особливо якщо використовуються нитки.

Shift Library

WiringPi включає в себе просту бібліотеку зсуву. Це дозволяє перенести 8-бітні значення даних з Pi або в Pi з таких пристроїв, як регістри зсуву (наприклад, 74HC595) тощо, хоча вони також можуть бути використані в деяких сценаріях розбиття бітів.

Для використання потрібно переконатися, що ваша програма містить такі файли:

```
#include <wiringPi.h>
```

```
#include <wiringShift.h>
```

Тоді доступні наступні дві функції:

uint8_t shiftIn (uint8_t dPin, uint8_t cPin, uint8_t order); зсуває 8-бітове значення біти, що з'являються на dPin у вихідну змінну, при цьому обмін синхронізується сигналом cPin . Порядок зсуву задається змінною order, що приймає значення LSBFIRST або MSBFIRST . Дані вибірки відбираються після того, як cPin переходить у 1.

void shiftOut (uint8_t dPin, uint8_t cPin, uint8_t order, uint8_t val); Зсув 8-бітового значення даних val, дані передаються через лінію на dPin а тактовий сигнал через cPin. порядок даних визначається змінною order. Дані можуть бути зафіксовані за будь-яким перепадом сигналу на лінії cPin. Зміна даних на лінії dPin здійснюється при низькому значенні на лінії cPin, а потім формується імпульс на лінії cPin і так повторюється для всіх 8 біт.

Програмна генерація сигналів

WiringPi включає в себе програмне забезпечення, здатне виводити простий тональний сигнал прямокутної форми на будь-який з GPIO-контактів Raspberry Pi. Однак існують деякі обмеження. Для підтримки низького використання процесора мінімальна ширина імпульсу становить 100 мкс. Це дає максимальну частоту 1 / 0,0002 = 5000 Гц. Також зауважте, що хоча root виконуються з більш високим пріоритетом у режимі реального часу, Linux все ще може впливати на точність генерованого тону.

Однак у цих обмеженнях можливий вивід звукових сигналів на динамік з високим імпедансом або п'єзовипромінювач.

Використовувати:

```
#include <wiringPi.h>
#include <softTone.h>
```

При компіляції програми необхідно включити бібліотеку PTHREAD, а також wiringPi бібліотеки:

```
cc -o myprog myprog.c -lwiringPi -lpthread
```

Необхідно ініціалізувати wiringPi за допомогою однієї з функцій wiringPiSetup(), wiringPiSetupGpio() або wiringPiSetupPhys (). Варіант wiringPiSetupSys () недостатньо швидкий, тому потрібно запускати свої програми з sudo.

Деякі модулі розширення також можуть бути досить швидкими для обробки програмного забезпечення ШІМ - це було протестовано, наприклад, з розширенням GPP MCP23S17 на PiFace.

Доступні наступні дві функції:

int softToneCreate (int pin); створює програмний керований тоновий конктакт. Ви можете використовувати будь-який контакт GPIO, і нумерація контактів буде такою, яку використовувала функція wiringPiSetup (). Повернене значення - 0 для успіху. Все інше, і вам слід перевірити глобальну змінну errno, щоб побачити, що пішло не так.

void softToneWrite (int pin, int freq); оновити значення частоти тону на даному контакті. Тон буде відтворюватися, поки частоту не встановити на 0.

Кожен контакт, активований у режимі softTone, використовує приблизно 0,5% ЦП. Необхідно утримувати програму у робочому стані для підтримки передачі звуку

Приклад простої програми роботи з ШІМ

Апаратний ШІМ на першому контакті

```
/* pwm.c:  */
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
int main (void)
{ int bright ;
  printf ("Raspberry Pi wiringPi PWM test program\n") ;
  if (wiringPiSetup () == -1)
    exit (1) ;
```

```
  pinMode (1, PWM_OUTPUT) ;
  for (;;)
  { for (bright = 0 ; bright < 1024 ; ++bright)
    { pwmWrite (1, bright) ; delay (1) ;   }
    for (bright = 1023 ; bright >= 0 ; --bright)
    { pwmWrite (1, bright) ; delay (1) ; }
  }
  return 0 ;
}
```

Програмний ШІМ на декількох контактах

```
/* * softPwm.c  */
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <wiringPi.h>
#include <softPwm.h>
#define RANGE  100
#define NUM_LEDS   8
int ledMap [NUM_LEDS] = { 0, 1, 2, 3, 4, 5, 6, 7 } ;
int values [NUM_LEDS] = { 0, 25, 50, 75, 100, 75, 50, 25 };
int main ()
{ int i, j ;
  char buf [80] ;
  wiringPiSetup ()  ;
  for (i = 0 ; i < NUM_LEDS ; ++i)
  { softPwmCreate (ledMap [i], 0, RANGE) ;
    printf ("%3d, %3d, %3d\n", i, ledMap [i], values [i]) ;
   }
  fgets (buf, 80, stdin) ;
// Bring all up one by one:
  for (i = 0 ; i < NUM_LEDS ; ++i)
    for (j = 0 ; j <= 100 ; ++j)
    { softPwmWrite (ledMap [i], j) ;  delay (10); }
  fgets (buf, 80, stdin) ;// All Down
  for (i = 100 ; i > 0 ; --i)
  { for (j = 0 ; j < NUM_LEDS ; ++j)
      softPwmWrite (ledMap [j], i) ; delay (10) ;
  }
  fgets (buf, 80, stdin) ;
  for (;;)
  {  for (i = 0 ; i < NUM_LEDS ; ++i)
      softPwmWrite (ledMap [i], values [i]) ;
    delay (50) ; i = values [0] ;
    for (j = 0 ; j < NUM_LEDS - 1 ; ++j)
      values [j] = values [j + 1] ;
    values [NUM_LEDS - 1] = i ;
  }
}
```

Програма для простого виводу даних на один із сегментів

Програма запитує код позиції та код для виводу і виводить обидва значення через зсувний регістр на підключені до них світлодіоди.

```
#include <stdio.h>
#include <wiringPi.h>
#include <wiringShift.h>
```

```
#define DPIN 6
#define CPIN 5
#define LPIN 2
int Pos = 0;
int Val = 0;
inline void OutData(int Val,int Pos)
{   digitalWrite(LPIN,0);
    shiftOut(DPIN,CPIN,1,Val);
    shiftOut(DPIN,CPIN,1,Pos);
    digitalWrite(LPIN,1);}
inline void DispStart()
{  pinMode (DPIN, OUTPUT);pullUpDnControl(DPIN,PUD_OFF);
   pinMode (CPIN, OUTPUT);pullUpDnControl(CPIN,PUD_OFF);
   pinMode (LPIN, OUTPUT);pullUpDnControl(LPIN,PUD_OFF); }
inline void DispStop()
{  pinMode (DPIN, INPUT);
   pinMode (CPIN, INPUT);
   pinMode (LPIN, INPUT);}
int main (void)
{ printf ("Raspberry Pi display test\n") ;
  wiringPiSetup () ;
  DispStart();
  for (;;) {
    printf("Set Position\n"); scanf("%i",&Pos);
    if(Pos==-1) break;
    printf("Set Value\n");   scanf("%i",&Val);
    OutData(Val,Pos); }
  digitalWrite(LPIN,0);
  shiftOut(DPIN,CPIN,1,0);
  digitalWrite(LPIN,1);
  DispStop();
  return 0 ;}
```

Програма виводу даних на дисплей

Програма зчитує дані з клавіатури та виводить їх на дисплей. Для можливості одночасної роботи вводу і виводу даних у динамічному режимі на дисплей змінюється режим роботи термінала.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <termios.h>
#include <stdint.h>
#include <sys/ioctl.h>
#include <wiringPi.h>
#include <wiringShift.h>
int charsWaiting (int fd) //визначення числа символів у буфері
{  int count ;
  if (ioctl (fd, FIONREAD, &count) == -1)
  { perror ("Something went wrong") ; exit (EXIT_FAILURE) ;  }
  return count ;}
#define DPIN 6
#define CPIN 5
#define LPIN 2
char DispBuff[4];
```

```
char PosToVal[4] ={1,2,4,8};
uint8_t DispPos=0;
void OutData()//вивід даних на дисплей
{   DispPos=(DispPos+1)&0x3;
    digitalWrite(LPIN,0);
    shiftOut(DPIN,CPIN,1,(DispBuff[DispPos]));
    shiftOut(DPIN,CPIN,1,(PosToVal[DispPos]));
    digitalWrite(LPIN,1);
};
void IntToDisp(int x)
{ char IntToVal[10] =
       {0xC0,0xf9,0xA4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
  if(x<0) x=-x;
  for(int i=4;i>0;)
  { DispBuff[--i]= IntToVal[x%10];
    x = x/10; }
};
void DispStart()
{ pinMode (DPIN, OUTPUT);pullUpDnControl(DPIN,PUD_OFF);
  pinMode (CPIN, OUTPUT);pullUpDnControl(CPIN,PUD_OFF);
  pinMode (LPIN, OUTPUT);pullUpDnControl(LPIN,PUD_OFF); };
void DispStop()
{ digitalWrite(LPIN,0);
  shiftOut(DPIN,CPIN,1,0);
  shiftOut(DPIN,CPIN,1,0);
  digitalWrite(LPIN,1);
  pinMode (DPIN, INPUT);
  pinMode (CPIN, INPUT);
  pinMode (LPIN, INPUT);
};
struct termios old, new ;
void TermStart()
{  tcgetattr (fileno (stdin), &old);
   tcgetattr (fileno (stdin), &new);
   new.c_oflag&=~OPOST;
   new.c_lflag&=~(ICANON|IEXTEN);//виключаємо пострічковий ввід
   tcsetattr (fileno (stdin), TCSANOW, &new) ;
};
void TermStop()
{ tcsetattr (fileno (stdin), TCSANOW, &old);};
char   exitflag = 1;
char   buff[80];
int8_t  buffpos=0;
int     val=0;
int    count;
int    main (void)
{  wiringPiSetup();
   DispStart();
   TermStart();
   printf("Enter a number or press Esc for exit\r\n");
   while( exitflag!=0)
   {OutData();
    usleep(10000);
    count = charsWaiting (fileno (stdin));
    if(count!=0)
```

```
    { char c=getchar();
       switch (c)  {
         case 0x1b: exitflag=0;break;
         case 0x7F: if(buffpos>0){buffpos--;};
            break;
         case 0xA:
            buff[buffpos]=0x0; buffpos=0;
            if(sscanf(buff,"%d",&val)!=1)
                  {printf("\rSome wrong...\n");}
            else
                  {printf("\rYou are wrote %d\r\n",val);
                   IntToDisp(val);}
            break;
         default:
            { buff[buffpos] = c;buffpos++;
              if(buffpos>78) buffpos=78;}
       }
     }
   }
   TermStop();  DispStop();printf("All done\n");
   return 0 ;}
```

Завдання на лабораторну роботу

1 Завантажити та зібрати програми, наведені у прикладі.

2 Модифікувати програми таким чином, щоб змінилась реакція на переривання.

3 Поєднайте програми щоб можна було паралельно виконувати програмний та апаратний ШІМ..

4 Спробуйте при запущеній програмі, що працює з апаратним та програмним ШІМ запустити якусь відносно тяжку програму. Спостерігайте за зміною сигналу ШІМ. Зробіть висновок.

5 Складіть звіт з лабораторної роботи.

Контрольні запитання

1. Для чого функції зсуву і якіі дії вони виконують?
2. Які фіункції ШІМ ви знаете?
3. Що таке режим ШІМ та як його ввімкнути?
4. Які часові функції можна використовувати при роботі з WiringPi?

Література

1. Иго Т. Arduino, датчики и сети для связи устройств: Пер. с англ. -СПб.: БХВ-Петербург, 2016. - 544 с.
2. Петин В.А. Arduino и Raspberry Pi в проектах Internet of Things. -СПб.: БХВ-Петербург, 2016.-464 с.
3. Петин В.А. Микрокомпьютеры Raspberry Pi.Практическое руководство. СПб.: БХВ-Петербург, 2015. -240 с.
4. Петин В.А. Проекты с использованием контроллера Arduino - СПб.: БХВ-Петербург, 2016. - 464 с.

Зміст