Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное
учреждение высшего образования
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Инженерно-технологическая академия ЮФУ
Институт компьютерных технологий и информационной безопасности

ЛАБОРАТОРНАЯ РАБОТА № 9
по дисциплине: «Введение в инженерную деятельность»

Тема: «Подключение и использование внешних стендов. Конвейер»

группа КТбо1-4

Выполнил
студент группы КТбо1-4 _____ А. А. Воронов

Принял
Ассистент ИКТИБ _____ А.А. Зубкова

Таганрог 2024

Содержание

1. Введение

Целью работы является изучение различных датчиков на внешних стендах.

Задачами работы являются:

1) Написание программ для выполнения различных сценариев;

2) Подключение к внешним цифровым и аналоговым устройствам с помощью блока Ports.

2. Основная часть

В рамках данной лабораторной работы рассматриваются стенды для выполнения различных заданий.

Для выполнения работы, требуется выполнить следующий ряд задач:

1) Включить датчик цвета;
2) Подключить сервоприводы;
3) Подключить шаговый двигатель;
4) Объединить всё в конвейер.

Согласно методическим указаниям, для объединения микроконтроллера и внешнего стенда с конвейером необходимо подключить общую землю (GND). Для работы с внешними устройства используется блок Ports. Для включения реле (вентиляторы, светодиодная лента, помпа) необходимо подавать высокий сигнал на порт PF.

Помимо этого, ещё были сделаны следующие пункты:

1) Прописана функция для установки альтернативного режима;
2) Настроен порт I2C1;
3) Настроен датчик цвета TCS34725;
4) Прописано поведение шагового двигателя;
5) Настроен сервопривод.

Программый код программы представлен на рисунках 1-6.

```
#include <string.h>
unsigned char SERVO_Sort=false;
unsigned short SERVO_New[4]={0,0,0,0};
unsigned short SERVO_Pos[4];

#define TCS34725_ADDRESS          (0x29)
#define TCS34725_COMMAND_BIT      (0x80)
#define TCS34725_ENABLE           (0x00)
#define TCS34725_ENABLE_AIEN      (0x10)    /* RGBC Interrupt Enable */
#define TCS34725_ENABLE_WEN       (0x08)    /* Wait enable - Writing 1 activates the wait timer */
#define TCS34725_ENABLE_AEN       (0x02)    /* RGBC Enable - Writing 1 actives the ADC, 0 disables it */
#define TCS34725_ENABLE_PON       (0x01)    /* Power on - Writing 1 activates the internal oscillator, 0 disables it */
#define TCS34725_ATIME            (0x01)    /* Integration time */
#define TCS34725_WTIME            (0x03)    /* Wait time (if TCS34725_ENABLE_WEN is asserted) */
#define TCS34725_WTIME_2_4MS      (0xFF)    /* WLONG0 = 2.4ms   WLONG1 = 0.029s */
#define TCS34725_WTIME_204MS      (0xAB)    /* WLONG0 = 204ms   WLONG1 = 2.45s  */
#define TCS34725_WTIME_614MS      (0x00)    /* WLONG0 = 614ms   WLONG1 = 7.4s   */
#define TCS34725_AILTL            (0x04)    /* Clear channel lower interrupt threshold */
#define TCS34725_AILTH            (0x05)
#define TCS34725_AIHTL            (0x06)    /* Clear channel upper interrupt threshold */
#define TCS34725_AIHTH            (0x07)
#define TCS34725_PERS             (0x0C)    /* Persistence register - basic SW filtering mechanism for interrupts */
#define TCS34725_PERS_NONE        (0b0000)  /* Every RGBC cycle generates an interrupt                            */
#define TCS34725_PERS_1_CYCLE     (0b0001)  /* 1 clean channel value outside threshold range generates an interrupt   */
#define TCS34725_PERS_2_CYCLE     (0b0010)  /* 2 clean channel values outside threshold range generates an interrupt  */
#define TCS34725_PERS_3_CYCLE     (0b0011)  /* 3 clean channel values outside threshold range generates an interrupt  */
#define TCS34725_PERS_5_CYCLE     (0b0100)  /* 5 clean channel values outside threshold range generates an interrupt  */
#define TCS34725_PERS_10_CYCLE    (0b0101)  /* 10 clean channel values outside threshold range generates an interrupt */
#define TCS34725_PERS_15_CYCLE    (0b0110)  /* 15 clean channel values outside threshold range generates an interrupt */
#define TCS34725_PERS_20_CYCLE    (0b0111)  /* 20 clean channel values outside threshold range generates an interrupt */
#define TCS34725_PERS_25_CYCLE    (0b1000)  /* 25 clean channel values outside threshold range generates an interrupt */
#define TCS34725_PERS_30_CYCLE    (0b1001)  /* 30 clean channel values outside threshold range generates an interrupt */
#define TCS34725_PERS_35_CYCLE    (0b1010)  /* 35 clean channel values outside threshold range generates an interrupt */
#define TCS34725_PERS_40_CYCLE    (0b1011)  /* 40 clean channel values outside threshold range generates an interrupt */
#define TCS34725_PERS_45_CYCLE    (0b1100)  /* 45 clean channel values outside threshold range generates an interrupt */
#define TCS34725_PERS_50_CYCLE    (0b1101)  /* 50 clean channel values outside threshold range generates an interrupt */
#define TCS34725_PERS_55_CYCLE    (0b1110)  /* 55 clean channel values outside threshold range generates an interrupt */
#define TCS34725_PERS_60_CYCLE    (0b1111)  /* 60 clean channel values outside threshold range generates an interrupt */
#define TCS34725_CONFIG           (0x0D)
#define TCS34725_CONFIG_WLONG     (0x02)    /* Choose between short and long (12x) wait times via TCS34725_WTIME */
#define TCS34725_CONTROL          (0x0F)    /* Set the gain level for the sensor */
#define TCS34725_ID               (0x12)    /* 0x44 = TCS34721/TCS34725, 0x4D = TCS34723/TCS34727 */
#define TCS34725_STATUS           (0x13)
#define TCS34725_STATUS_AINT      (0x10)    /* RGBC Clean channel interrupt */
#define TCS34725_STATUS_AVALID    (0x01)    /* Indicates that the RGBC channels have completed an integration cycle */
#define TCS34725_CDATAL           (0x14)    /* Clear channel data */
#define TCS34725_CDATAH           (0x15)
#define TCS34725_RDATAL           (0x16)    /* Red channel data */
#define TCS34725_RDATAH           (0x17)
#define TCS34725_GDATAL           (0x18)    /* Green channel data */
#define TCS34725_GDATAH           (0x19)
#define TCS34725_BDATAL           (0x1A)    /* Blue channel data */
#define TCS34725_BDATAH           (0x1B)

typedef enum {
  TCS34725_INTEGRATIONTIME_2_4MS  = 0xFF,   /**<  2.4ms - 1 cycle    - Max Count: 1024  */
  TCS34725_INTEGRATIONTIME_24MS   = 0xF6,   /**<  24ms  - 10 cycles  - Max Count: 10240 */
  TCS34725_INTEGRATIONTIME_50MS   = 0xEB,   /**<  50ms  - 20 cycles  - Max Count: 20480 */
  TCS34725_INTEGRATIONTIME_101MS  = 0xD5,   /**<  101ms - 42 cycles  - Max Count: 43008 */
  TCS34725_INTEGRATIONTIME_154MS  = 0xC0,   /**<  154ms - 64 cycles  - Max Count: 65535 */
  TCS34725_INTEGRATIONTIME_700MS  = 0x00    /**<  700ms - 256 cycles - Max Count: 65535 */
} tcs34725IntegrationTime_t;

typedef enum {
  TCS34725_GAIN_1X              = 0x00,   /**<  No gain  */
  TCS34725_GAIN_4X              = 0x01,   /**<  4x gain  */
  TCS34725_GAIN_16X             = 0x02,   /**<  16x gain */
  TCS34725_GAIN_60X             = 0x03    /**<  60x gain */
} tcs34725Gain_t;

struct {
  unsigned short Clear;
  unsigned short Red;
  unsigned short Green;
  unsigned short Blue;
} LRGB;

int clear; // Общая интенсивность светового потока
int red; // Интенсивность красного цвета
int green; // Интенсивность зелёного цвета
int blue; // Интенсивность синего цвета

char reg[2];
```

Рисунок 1 - Определение каналов

```
char reg[2];

void colorSensor(){

  reg[0]=TCS34725_ATIME; reg[1]=TCS34725_INTEGRATIONTIME_50MS;
  I2C_Write(TCS34725_ADDRESS, reg, 2);

  reg[0]=TCS34725_CONTROL; reg[1]=TCS34725_GAIN_4X;
  I2C_Write(TCS34725_ADDRESS, reg, 2);

  reg[0]=TCS34725_ENABLE; reg[1]=TCS34725_ENABLE_PON;
  I2C_Write(TCS34725_ADDRESS, reg, 2);
  for(int a=0; a<100000; a++);
  reg[0]=TCS34725_ENABLE; reg[1]=TCS34725_ENABLE_PON | TCS34725_ENABLE_AEN;
  I2C_Write(TCS34725_ADDRESS, reg, 2);

  reg[0]=TCS34725_ENABLE; reg[1]=TCS34725_ENABLE_PON | TCS34725_ENABLE_AEN | TCS34725_ENABLE_AIEN;
    I2C_Write(TCS34725_ADDRESS, reg, 2);

    reg[0]=TCS34725_COMMAND_BIT | TCS34725_CDATAL;
    I2C_Write(TCS34725_ADDRESS, reg, 1);

    I2C_Read(TCS34725_ADDRESS, (char*)&LRGB, 8); // Получим данные из устройства

    for(int a=0; a<100000; a++);

    reg[0]=TCS34725_ENABLE; reg[1]=TCS34725_ENABLE_PON | TCS34725_ENABLE_AEN;
    I2C_Write(TCS34725_ADDRESS, reg, 2);

    for(int a=0; a<100000; a++);
}

int delay = 40000;
void Stepper(){
   GPIOF->BSRRL = GPIO_BSRR_BS_11; // Установить значение HIGH для IN1
    GPIOF->BSRRL = GPIO_BSRR_BS_2; // Установить значение HIGH для IN2
    GPIOF->BSRRH = GPIO_BSRR_BS_1; // Установить значение LOW для IN3
    GPIOF->BSRRH = GPIO_BSRR_BS_0; // Установить значение LOW для IN4
      for(int a=0; a<delay; a++);
     GPIOF->BSRRH = GPIO_BSRR_BS_11; // Установить значение LOW для IN1
    GPIOF->BSRRL = GPIO_BSRR_BS_2; // Установить значение HIGH (для IN2
    GPIOF->BSRRL = GPIO_BSRR_BS_1; // Установить значение HIGH для IN3
    GPIOF->BSRRH = GPIO_BSRR_BS_0; // Установить значение LOW для IN4
      for(int a=0; a<delay; a++);
    GPIOF->BSRRH = GPIO_BSRR_BS_11; // Установить значение LOW для IN1
    GPIOF->BSRRH = GPIO_BSRR_BS_2; // Установить значение LOW для IN2
    GPIOF->BSRRL = GPIO_BSRR_BS_1; // Установить значение HIGH для IN3
    GPIOF->BSRRL = GPIO_BSRR_BS_0; // Установить значение HIGH для IN4
      for(int a=0; a<delay; a++);
    GPIOF->BSRRL = GPIO_BSRR_BS_11; // Установить значение HIGH для IN1
    GPIOF->BSRRH = GPIO_BSRR_BS_2; // Установить значение LOW для IN2
    GPIOF->BSRRH = GPIO_BSRR_BS_1; // Установить значение LOW для IN3
    GPIOF->BSRRL = GPIO_BSRR_BS_0; // Установить значение HIGH для IN4
      for(int a=0; a<delay; a++);
}
```

Рисунок 2 - Настройка шагового двигателя

```
extern "C" void TIM8_TRG_COM_TIM14_IRQHandler()
{
  unsigned short state=TIM14->SR;
  TIM14->SR=0;
  //---
  unsigned short next=0xFFFF;
  //---
  if(state & TIM_SR_UIF)
  {
    if(!SERVO_Sort)
    {
      memcpy(SERVO_Pos, SERVO_New, sizeof(SERVO_New));
      SERVO_Sort=true;
    }
    //---
    if(SERVO_Pos[0])  { GPIOF->BSRRL = 1<<12;  if(SERVO_Pos[0]<next)  next=SERVO_Pos[0];  }
    if(SERVO_Pos[1])  { GPIOF->BSRRL = 1<<13;  if(SERVO_Pos[1]<next)  next=SERVO_Pos[1];  }
    if(SERVO_Pos[2])  { GPIOF->BSRRL = 1<<14;  if(SERVO_Pos[2]<next)  next=SERVO_Pos[2];  }
    if(SERVO_Pos[3])  { GPIOF->BSRRL = 1<<15;  if(SERVO_Pos[3]<next)  next=SERVO_Pos[3];  }
  }
  else
  {
    unsigned short timer=TIM14->CNT;
    //---
    if(SERVO_Pos[0])  { if(SERVO_Pos[0]<=timer)  GPIOF->BSRRH = 1<<12;   else if(SERVO_Pos[0]<next) next=SERVO_Pos[0]; }
    if(SERVO_Pos[1])  { if(SERVO_Pos[1]<=timer)  GPIOF->BSRRH = 1<<13;   else if(SERVO_Pos[1]<next) next=SERVO_Pos[1]; }
    if(SERVO_Pos[2])  { if(SERVO_Pos[2]<=timer)  GPIOF->BSRRH = 1<<14;   else if(SERVO_Pos[2]<next) next=SERVO_Pos[2]; }
    if(SERVO_Pos[3])  { if(SERVO_Pos[3]<=timer)  GPIOF->BSRRH = 1<<15;   else if(SERVO_Pos[3]<next) next=SERVO_Pos[3]; }
  }
  //---
  if(next!=0xFFFF)
  {
    TIM14->CR1 = 0;
    //---
    if(next<=TIM14->CNT) TIM14->CCR1=TIM14->CNT+2; else TIM14->CCR1=next;
    //---
    TIM14->CR1 = TIM_CR1_CEN;
  }
}

void SERVO_Init()
{
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOFEN;
  //---
  GPIOF->MODER &= ~(GPIO_MODER_MODER12 | GPIO_MODER_MODER13 | GPIO_MODER_MODER14 | GPIO_MODER_MODER15);
  GPIOF->MODER |= GPIO_MODER_MODER12_0 | GPIO_MODER_MODER13_0 | GPIO_MODER_MODER14_0 | GPIO_MODER_MODER15_0;
  //---
  NVIC_SetPriority(TIM8_TRG_COM_TIM14_IRQn, 2);
  NVIC_EnableIRQ(TIM8_TRG_COM_TIM14_IRQn);
  //---
  RCC->APB1ENR |= RCC_APB1ENR_TIM14EN;
  //---
  TIM14->CR1  = 0;
  TIM14->CNT  = 0;
  TIM14->PSC  = ((SystemCoreClock/2)/50/20000)-1;
  TIM14->ARR  = 20000;
  TIM14->CR2  = TIM_CR2_MMS_1 | TIM_CR2_MMS_0;
  TIM14->DIER = TIM_DIER_UIE | TIM_DIER_CC1IE;
  TIM14->BDTR = 0;
  //---
  TIM14->CCMR1 = 0;
  TIM14->CCMR2 = 0;
  TIM14->CCER  = 0;
  //---
  TIM14->CCR1 = 0;
  //---
  TIM14->CR1 = TIM_CR1_CEN;
}

void SERVO_Set(int I_Servo, int I_Value)
{
  if(I_Servo>4) return;
  //---
  if(I_Value==0xFFFF) return;
  //---
  SERVO_Sort=false;
  //---
  SERVO_New[I_Servo]=I_Value;
}
```

Рисунок 3 - Настройка сервоприводов

```
218
219 void SERVO_Open_close_Conveyor(int Select_conveyor, int Select_Servo, bool state)
220 {
221   // Для конвейера №1
222   if(Select_conveyor == 1){
223     switch(Select_Servo){
224     case 0:
225       // Сервопривод №1 от датчика цвета
226       if(Select_Servo == 0 and state == true)
227       {
228         SERVO_Set(Select_Servo, 1274);
229       }
230       else {
231         SERVO_Set(Select_Servo, 2400);
232       }
233       break;
234     case 1:
235       // Сервопривод №2 от датчика цвета
236       if(Select_Servo == 1 and state == true)
237       {
238         SERVO_Set(Select_Servo, 1224);
239       }
240       else {
241         SERVO_Set(Select_Servo, 2400);
242       }
243       break;
244     case 2:
245       // Сервопривод №3 от датчика цвета
246       if(Select_Servo == 2 and state == true)
247       {
248         SERVO_Set(Select_Servo, 1075);
249       }
250       else {
251         SERVO_Set(Select_Servo, 2320);
252       }
253       break;
254     }
255   }
256   // Для конвейера №2
257   if(Select_conveyor == 2){
258     switch(Select_Servo){
259     case 0:
260       // Сервопривод №1 от датчика цвета
261       if(Select_Servo == 0 and state == true)
262       {
263         SERVO_Set(Select_Servo, 1148);
264       }
265       else {
266         SERVO_Set(Select_Servo, 2285);
267       }
268       break;
269     case 1:
270       // Сервопривод №2 от датчика цвета
271       if(Select_Servo == 1 and state == true)
272       {
273         SERVO_Set(Select_Servo, 1100);
274       }
275       else {
276         SERVO_Set(Select_Servo, 2200);
277       }
278       break;
279     case 2:
280       // Сервопривод №3 от датчика цвета
281       if(Select_Servo == 2 and state == true)
282       {
283         SERVO_Set(Select_Servo, 1095);
284       }
285       else {
286         SERVO_Set(Select_Servo, 2340);
287       }
288       break;
289     }
290   }
291 }
```

Рисунок 4 - Функции для работы с сервоприводом

```
295 void SetAltFunc(GPIO_TypeDef* Port, int Channel, int AF)
296 {
297   Port->MODER &= ~(3<<(2*Channel)); // Сброс режима
298   Port->MODER |= 2<<(2*Channel); // Установка альт. Режима
299 
300   if(Channel<8) // Выбор регистра зависит от номера контакта
301   {
302     Port->AFR[0] &= ~(15<<4*Channel); // Сброс альт. функции
303     Port->AFR[0] |= AF<<(4*Channel); // Установка альт. функции
304   }
305   else
306   {
307     Port->AFR[1] &= ~(15<<4*(Channel-8)); // Сброс альт. функции
308     Port->AFR[1] |= AF<<(4*(Channel-8)); // Установка альт. функции
309   }
310 }
311 
312 void I2C_Read(int Address, char* Data, int Size) // Функция приёма
313 {
314   if(Size>1) I2C1->CR1 |= I2C_CR1_ACK; // Прём больше 1 байт то нужен ACK
315   else I2C1->CR1 &= ~I2C_CR1_ACK; // Прём меньше 2 байт то ACK не нужен
316   I2C1->CR1 |= I2C_CR1_START; // Занимаем линию связи для данных
317   while (!(I2C1->SR1 & I2C_SR1_SB)) { } // Ждём занятия линии
318   I2C1->DR = (Address<<1) | I2C_OAR1_ADD0; // Шлём адрес для приёма данных
319   while (!(I2C1->SR1 & I2C_SR1_ADDR)) { } // Ждём успешной связи с устройством
320   I2C1->SR2; // Читаем SR2 для его отчистки
321   while (Size--) // Цикл приёма байт
322   {
323     while (!(I2C1->SR1 & I2C_SR1_RXNE)) { } // Ждем наличия байта для чтения
324     if(Size == 1) I2C1->CR1 &= ~I2C_CR1_ACK; // Если остался 1 байт, то ACK убрать
325     *Data++ = I2C1->DR; // Читаем пришедший байт
326   }
327   I2C1->CR1 |= I2C_CR1_STOP; // Освобождаем линию связи
328   while (I2C1->CR1 & I2C_CR1_STOP) { } // Ждём освобождения линии
329 }
330 
331 void I2C_Write(int Address, char* Data, int Size) // Функция передачи
332 {
333   I2C1->CR1 |= I2C_CR1_START; // Занимаем линию связи для данных
334   while (!(I2C1->SR1 & I2C_SR1_SB)) { } // Ждём занятия линии
335   I2C1->DR = (Address<<1) & ~I2C_OAR1_ADD0; // Шлём адрес для передачи данных
336   while (!(I2C1->SR1 & I2C_SR1_ADDR)) { } // Ждём успешной связи с устройством
337   I2C1->SR2; // Читаем SR2 для его отчистки
338   while (Size--) // Цикл передачи байт
339   {
340     while (!(I2C1->SR1 & I2C_SR1_TXE)) { } // Ждем готовности передать байт
341     I2C1->DR = *Data++; // Передаём байт
342   }
343   while (!(I2C1->SR1 & I2C_SR1_BTF)) { } // Ждём окончания передачи данных
344   I2C1->CR1 |= I2C_CR1_STOP; // Освобождаем линию связи
345   while (I2C1->CR1 & I2C_CR1_STOP) { } // Ждём освобождения линии
346 }
```

Рисунок 5 - Функция для установки альтернативного режима

```
348
349 Int main()
350 {
351     RCC->AHB1ENR |= RCC_AHB1ENR_GPIOFEN; // Порт F задействован
352     GPIOF->MODER &= ~(GPIO_MODER_MODER0 | GPIO_MODER_MODER1 | GPIO_MODER_MODER2 | GPIO_MODER_MODER3 | GPIO_MODER_MODER4
        | GPIO_MODER_MODER12); // Сброс режима
353     GPIOF->MODER |= GPIO_MODER_MODER12_0; // Установка режима на выход
354     GPIOF->BSRRL = 1<<12;
355
356     RCC->APB1ENR |= RCC_APB1ENR_I2C1EN; // I2C 1 задействован
357     RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN; // Порт B задействован
358     GPIOB->OTYPER |= GPIO_OTYPER_OT_9 | GPIO_OTYPER_OT_8; // Открытый сток для PB9 и PB8
359     GPIOB->OSPEEDR |= GPIO_OSPEEDER_OSPEEDR9_0 | GPIO_OSPEEDER_OSPEEDR9_1; // Max. скорость PB9
360     GPIOB->OSPEEDR |= GPIO_OSPEEDER_OSPEEDR8_0 | GPIO_OSPEEDER_OSPEEDR8_1; // Max. скорость PB8
361     GPIOB->PUPDR |= GPIO_PUPDR_PUPDR9_0 | GPIO_PUPDR_PUPDR8_0; // Подтяжка 3.3V для PB9 и PB8
362     SetAltFunc(GPIOB, 8, 4); // Установка альт. режима AF4 для SCL(PB8)
363     SetAltFunc(GPIOB, 9, 4); // Установка альт. режима AF4 для SDA(PB9)
364
365     I2C1->CR2 = (I2C_CR2_FREQ & 0x2A);
366     I2C1->CCR = I2C_CCR_FS | (I2C_CCR_CCR & 0x006C);
367     I2C1->TRISE = (I2C_TRISE_TRISE & 0x14);
368
369     I2C1->CR1 = I2C_CR1_PE; // Запуск I2C устройства
370
371     while(I2C1->SR2 & I2C_SR2_BUSY) { } // Ожидание готовности устройства
372
373     while(1)
374     {
375       colorSensor();
376       clear = LRGB.Clear; // Общая интенсивность светового потока
377       red = LRGB.Red; // Интенсивность красного цвета
378       green = LRGB.Green; // Интенсивность зелёного цвета
379       blue = LRGB.Blue; // Интенсивность синего цвета
380     }
381
382     SERVO_Init();
383
384 while(){
385     SERVO_Open_close_Conveyor (1, 0, true);
386 }
387  }
```

Рисунок 6 - Главная функция программы

3. Вывод

В данной лабораторной работе были рассмотрены внешние стенды, подключаемые к микроконтроллеру. Была произведена настройка стенда с конвейером, написаны функции для выполнения различных сценариев и подключения к аналоговым и цифровым устройствам.