```
//-------------------------------------------------------------------------------
// Программа управления игрой ПВКК-2017 для платы ВСПИ.01.022.02
// Создана 21.08.2018
// Последнее измененеие 20.10.18
//-------------------------------------------------------------------------------
// Использование портов:
// Порт A - по шине I2C подключно ЭППЗУ для записи звуковых файлов
// Порт B:
// RB0 - вход кнопки "ШАГ"
// RB1 - не исп (выход=0)
// RB2 - выход PWM1 (звук)
// RB3 - выход гирлянда ЗЕЛЁНЫЙ. 1 - гирлянда горит
// RB4 - выход гирлянда КРАСНЫЙ. 1 - гирлянда горит
// RB5 - выход КОЗА    правый берег (3-4). 1 - горит
// RB6 - выход КАПУСТА правый берег (4-4). 1 - горит
// RB7 - выход РЕКА (5). 1 - горит
// Порт C:
// RC0 - выход сигнала RS на индикатор. 0 - инструкция, 1 - данные.
// RC1 - выход сигнала RW на индикатор. 0 - запись в контроллер, 1 - чтение из.
// RC2 - выход сигнала E на индикатор. 0 - нет работы, 1 - строб.
// RC4..7 - не исп. (выходы=0)
// Порт D: шина данных индикатора (ввод/вывод)
// Порт E:
// RE0 - вход кнопки ПЕРЕВОЗЧИК
// RE1 - вход кнопки ВОЛК
// RE2 - вход кнопки КОЗА
// RE3 - вход кнопки КАПУСТА
// Порт F:
// RF0: выход светодиод ПЕРЕВОЗЧИК левый берег (1-1) 1 - горит
// RF1: выход светодиод ВОЛК       левый берег (2-1)
// RF2: выход светодиод КОЗА       левый берег (3-1)
// RF3: выход светодиод КАПУСТА    левый берег (4-1)
// RF4: выход светодиод ПЕРЕВОЗЧИК левая лодка (1-2)
// RF5: выход светодиод ВОЛК       левая лодка (2-2)
// RF6: выход светодиод КОЗА       левая лодка (3-2)
// RF7: выход светодиод КАПУСТА    левая лодка (4-2)
// Порт G:
// RG0: выход светодиод ПЕРЕВОЗЧИК правая лодка (1-3) 1 - горит
// RG1: выход светодиод ВОЛК       правая лодка (2-3)
// RG2: выход светодиод КОЗА       правая лодка (3-3)
// RG3: выход светодиод КАПУСТА    правая лодка (4-3)
// RG4: выход светодиод ПЕРЕВОЗЧИК правый берег (1-4)
// RG5: выход светодиод ВОЛК       правый берег (2-4)
// RG6: вход (КИА-RX)
// RG7: выход (КИА-TX)
//-------------------------------------------------------------------------------
#include "1886VE2dcor.h"
#include "int1886cor.h"
#pragma origin 0x8
//-------------------------------------------------------------------------------
#define BLINK_MS 200
#define LINE1 0x80
#define LINE2 0xC0
#define MIDN 0x10000
#define AUDIO_1 20196
#define AUDIO_2 33468
#define AUDIO_3 43764
#define AUDIO_4 54354
#define AUDIO_5 65724
#define AUDIO_6 79632
#define NS 90808
//-------------------------------------------------------------------------------
// Список звуковых фрагментов:
// 0 - Помоги перевозчику переправить всех на другой берег реки!
// 1 - Молодец, задание выполнено!
// 2 - Волк съел козу!
// 3 - Коза съела капусту!
// 4 - В лодке только одно место!
// 5 - Без перевозчика лодка не поплывет!
// 6 - Лодка на другом берегу!
uns8 sPORTB, sPORTC, sPORTG, sPORTF;
uns8 Perev_pos, Volk_pos, Koza_pos, Kapusta_pos;// Позиции персонажей от 1 до 4
uns8 event; // Код логического события
bit KN_HOD @ PORTB.0;// Кнопка ХОД подключена к биту 0 порта B
bit KN_PEREV @ PORTE.0;// Кнопка ПЕРЕВОЗЧИК подключена к биту 0 порта E
bit KN_VOLK @ PORTE.1;// Кнопка ВОЛК подключена к биту 1 порта E
bit KN_KOZA @ PORTE.2;// Кнопка КОЗА подключена к биту 2 порта E
bit KN_KAPUSTA @ PORTE.3; // Кнопка КАПУСТА подключена к биту 3 порта E
```

```
bit BUSY_FLAG @ PORTD.7; //Сигнал занятости процессора индикатора
// Глобальные переменные для обработки прерываний
uns8 idiv, inum;
uns24 iptr, istop;
bit isAudioStart;
uns8 kia_stat, kia_cs;
uns24 nbuf, kia_count;
//-------------------------------------------------------------------------------
// Функция-обработчик прерываний:
//-------------------------------------------------------------------------------
interrupt iServer(void)
{
    multi_interrupt_entry_and_save
//-------------------------------------------------------------------------------
PERIPHERAL_service:
// Обработка прерываний от периферии:
  if (TMR1IF)  // Если это наше прерывание(30КГц)
  {
    // Работать будем на каждом 5 прерывании, чтобы получить частоту выдачи 6КГц
    if (idiv==4)
    {
      idiv=0;
      // Здесь нужно выдать на ШИМ новое значение:
      if (isAudioStart)
      {
        if (iptr<istop)
        {
          //-------------------------------------------------------------------
          // Включаем формирование признака СТАРТА на шине
          SEN=1; while (SSPIF==0); SSPIF=0;
          // Передаем адрес микросхемы для чтения (управляющий байт)
          if (iptr<MIDN) SSPBUF=0b10100001; else SSPBUF=0b10100011;
          while (SSPIF==0); SSPIF=0;
          // Считываем байт данных ячейки:
          RCEN=1; while (SSPIF==0); SSPIF=0;
          PW1DCH=SSPBUF;
          // Включаем формирование признака СТОП на шине
          PEN=1; while (SSPIF==0); SSPIF=0;
          //-------------------------------------------------------------------
          iptr++;
        } else { PW1DCH=100; isAudioStart=0;}// Воспроизведение закончено
      }
    } else idiv++;
    TMR1IF=0;// Флаг прерывания нужно убрать самим
  }
goto RESTORE_and_return;
//-------------------------------------------------------------------------------
TMR0_service:
  // Если надо, сохрани: PRODL,PRODH,TBLPTRH,TBLPTRL,FSR0,FSR1:
  // Обработчик прерываний уровня 2 (таймер0)
  // T0IF автоматически сбрасывается когда процессор переходит по вектору 0x10
  // Если сохранял, восстанови: PRODL,PRODH,TBLPTRH,TBLPTRL,FSR0,FSR1:
goto RESTORE_and_return;
//-------------------------------------------------------------------------------
T0CKI_service:
goto RESTORE_and_return;
//-------------------------------------------------------------------------------
 INT_service:
//-------------------------------------------------------------------------------
RESTORE_and_return:
    interrupt_exit_and_restore
}
//-------------------------------------------------------------------------------
void delay_timer3_ms(uns16 n)
// Задержка на n милисекунд с таймером 3
{
  PR3H=0x17;  PR3L=0x70;// Значение периода 1 ms = 6000
  TMR3CS=0;//Уст-ить внутренний такт FC/4
  TMR3IE=0;// Запретить прерывание от таймера
  TMR3ON=1;// Запускаем таймер
  while(n>0)
  {
    while(TMR3IF==0); // Ожидаем срабатывание таймера
    TMR3IF=0; n--;
  }
  TMR3ON=0;
}
//-------------------------------------------------------------------------------
```

```
157: uns8 read_rom_first_byte(uns24);
158: //-----------------------------------------------------------------------------
159: void start_audio(uns8 n)
160: {
161:  if (isAudioStart) return;
162:   switch (n)
163:   {
164:     case 0: iptr=0;       istop=AUDIO_1; break;
165:     case 1: iptr=AUDIO_1; istop=AUDIO_2; break;
166:     case 2: iptr=AUDIO_2; istop=AUDIO_3; break;
167:     case 3: iptr=AUDIO_3; istop=AUDIO_4; break;
168:     case 4: iptr=AUDIO_4; istop=AUDIO_5; break;
169:     case 5: iptr=AUDIO_5; istop=AUDIO_6; break;
170:     case 6: iptr=AUDIO_6; istop=NS; break;
171:   }
172:   read_rom_first_byte(iptr); iptr++;
173:   isAudioStart=1;
174: }
175: //-----------------------------------------------------------------------------
176: // Функции для работы с ЭППЗУ по I2C:
177: void wait_ssp_complete(void)
178: {
179:   while (SSPIF==0); // Ждем завершение операции без таймаута
180:   SSPIF=0;
181: }
182: //-----------------------------------------------------------------------------
183: uns8 read_rom_first_byte(uns24 adress) // 70мкс
184: {
185:  uns8 data;
186:
187:  SSPIF=0;
188:  // Включаем формирование признака СТАРТ на шине
189:  SEN=1;  wait_ssp_complete();
190:  // Передаем адрес микросхемы
191:  if (adress<MIDN) SSPBUF=0b10100000; else SSPBUF=0b10100010;  wait_ssp_complete();
192:  // Передаем старший байт адреса ячейки
193:  SSPBUF=adress.mid8;  wait_ssp_complete();
194:  // Передаем младший байт адреса ячейки
195:  SSPBUF=adress.low8;  wait_ssp_complete();
196:  // Включаем формирование признака повторный СТАРТ
197:  RSEN=1;  wait_ssp_complete();
198:  // Передаем адрес микросхемы для чтения
199:  if (adress<MIDN) SSPBUF=0b10100001; else SSPBUF=0b10100011;
200:  wait_ssp_complete();
201:  // Считываем байт данных ячейки
202:  RCEN=1;  wait_ssp_complete();
203:  data=SSPBUF;
204:  // Включаем формирование признака стоп на шине
205:  PEN=1;  wait_ssp_complete();
206:  return data;
207: }
208: //-----------------------------------------------------------------------------
209: void BlinkRed(uns8 n)
210: {
211:   uns8 i;
212:   for (i=0;i<n;i++)
213:   {
214:     sPORTB.4=1; PORTB=sPORTB; delay_timer3_ms(BLINK_MS);
215:     sPORTB.4=0; PORTB=sPORTB; delay_timer3_ms(BLINK_MS);
216:   }
217: }
218: //-----------------------------------------------------------------------------
219: void BlinkGreen(uns8 n)
220: {
221:   uns8 i;
222:   for (i=0;i<n;i++)
223:   {
224:     sPORTB.3=1; PORTB=sPORTB; delay_timer3_ms(BLINK_MS);
225:     sPORTB.3=0; PORTB=sPORTB; delay_timer3_ms(BLINK_MS);
226:   }
227: }
228: //-----------------------------------------------------------------------------
229: void BlinkRiver(uns8 n)
230: {
231:   uns8 i;
232:   for (i=0;i<n;i++)
233:   {
234:     sPORTB.7=1; PORTB=sPORTB; delay_timer3_ms(BLINK_MS);
```

```
235:     sPORTB.7=0; PORTB=sPORTB; delay_timer3_ms(BLINK_MS);
236:   }
237: }
238: //-----------------------------------------------------------------------------
239: void Perev_move(uns8 pos)// ПОместить перевозчика в позицию pos=1..4
240: {
241:   //Сначала нужно погасить светодиод в текущей позиции
242:   switch (Perev_pos)
243:   {
244:     case 1: sPORTF.0=0; PORTF=sPORTF; break;
245:     case 2: sPORTF.4=0; PORTF=sPORTF; break;
246:     case 3: sPORTG.0=0; PORTG=sPORTG; break;
247:     case 4: sPORTG.4=0; PORTG=sPORTG; break;
248:   }
249:   Perev_pos=pos; // Ставим в новую позицию
250:   // Теперь зажигаем светодиод в новой позиции
251:   switch (Perev_pos)
252:   {
253:     case 1: sPORTF.0=1; PORTF=sPORTF; break;
254:     case 2: sPORTF.4=1; PORTF=sPORTF; break;
255:     case 3: sPORTG.0=1; PORTG=sPORTG; break;
256:     case 4: sPORTG.4=1; PORTG=sPORTG; break;
257:   }
258: }
259: //-----------------------------------------------------------------------------
260: void Volk_move(uns8 pos)// Поместить волка в позицию pos=1..4
261: {
262:   //Сначала нужно погасить светодиод в текущей позиции
263:   switch (Volk_pos)
264:   {
265:     case 1: sPORTF.1=0; PORTF=sPORTF; break;
266:     case 2: sPORTF.5=0; PORTF=sPORTF; break;
267:     case 3: sPORTG.1=0; PORTG=sPORTG; break;
268:     case 4: sPORTG.5=0; PORTG=sPORTG; break;
269:   }
270:   Volk_pos=pos; // Ставим в новую позицию
271:   // Теперь зажигаем светодиод в новой позиции
272:   switch (Volk_pos)
273:   {
274:     case 1: sPORTF.1=1; PORTF=sPORTF; break;
275:     case 2: sPORTF.5=1; PORTF=sPORTF; break;
276:     case 3: sPORTG.1=1; PORTG=sPORTG; break;
277:     case 4: sPORTG.5=1; PORTG=sPORTG; break;
278:   }
279: }
280: //-----------------------------------------------------------------------------
281: void Koza_move(uns8 pos)// Поместить козу в позицию pos=1..4
282: {
283:   //Сначала нужно погасить светодиод в текущей позиции
284:   switch (Koza_pos)
285:   {
286:     case 1: sPORTF.2=0; PORTF=sPORTF; break;
287:     case 2: sPORTF.6=0; PORTF=sPORTF; break;
288:     case 3: sPORTG.2=0; PORTG=sPORTG; break;
289:     case 4: sPORTB.5=0; PORTB=sPORTB; break;
290:   }
291:   Koza_pos=pos; // Ставим в новую позицию
292:   // Теперь зажигаем светодиод в новой позиции
293:   switch (Koza_pos)
294:   {
295:     case 1: sPORTF.2=1; PORTF=sPORTF; break;
296:     case 2: sPORTF.6=1; PORTF=sPORTF; break;
297:     case 3: sPORTG.2=1; PORTG=sPORTG; break;
298:     case 4: sPORTB.5=1; PORTB=sPORTB; break;
299:   }
300: }
301: //-----------------------------------------------------------------------------
302: void Kapusta_move(uns8 pos)// Поместить капусту в позицию pos=1..4
303: {
304:   //Сначала нужно погасить светодиод в текущей позиции
305:   switch (Kapusta_pos)
306:   {
307:     case 1: sPORTF.3=0; PORTF=sPORTF; break;
308:     case 2: sPORTF.7=0; PORTF=sPORTF; break;
309:     case 3: sPORTG.3=0; PORTG=sPORTG; break;
310:     case 4: sPORTB.6=0; PORTB=sPORTB; break;
311:   }
312:   Kapusta_pos=pos; // Ставим в новую позицию
```

```
313:   // Теперь зажигаем светодиод в новой позиции
314:   switch (Kapusta_pos)
315:   {
316:     case 1: sPORTF.3=1; PORTF=sPORTF; break;
317:     case 2: sPORTF.7=1; PORTF=sPORTF; break;
318:     case 3: sPORTG.3=1; PORTG=sPORTG; break;
319:     case 4: sPORTB.6=1; PORTB=sPORTB; break;
320:   }
321: }
322: //-----------------------------------------------------------------------------
323: void KeyPressed_Perev(void) // Обработчик нажатий на кнопку Перевозчик
324: {
325:   switch (Perev_pos)
326:   {
327:     case 1: Perev_move(2);
328:             if ((Volk_pos==2)||(Koza_pos==2)||(Kapusta_pos==2)) event=4; //Лодка готова!
329:             break;
330:     case 2: Perev_move(1); break;
331:     case 3: Perev_move(4); break;
332:     case 4: Perev_move(3); break;
333:   }
334:
335: }
336:
337: //-----------------------------------------------------------------------------
338: void KeyPressed_Volk(void) // Обработчик нажатий на кнопку Волк
339: {
340:   if (Perev_pos<3) // Лодка на левом берегу
341:   {
342:     switch (Volk_pos)
343:     {
344:       case 1: if ((Koza_pos==2)||(Kapusta_pos==2)) event=1; // Событие 1 - лодка занята
345:               else
346:               {
347:                 Volk_move(2);
348:                 if (Perev_pos==2) event=4; //Лодка готова!
349:               }
350:               break;
351:       case 2: Volk_move(1); break;
352:       default: event=3; // Волк на другом берегу!
353:     }
354:   } else // Лодка на правом берегу
355:   {
356:     switch (Volk_pos)
357:     {
358:       case 3: Volk_move(4); break;
359:       case 4: if ((Koza_pos==3)||(Kapusta_pos==3)) event=1; // Событие 1 - лодка занята
360:               else
361:               {
362:                 Volk_move(3);
363:                 if (Perev_pos==3) event=4; //Лодка готова!
364:               }
365:               break;
366:       default: event=3; // Волк на другом берегу!
367:     }
368:   }
369: }
370: //-----------------------------------------------------------------------------
371: void KeyPressed_Koza(void) // Обработчик нажатий на кнопку Коза
372: {
373:   if (Perev_pos<3) // Лодка на левом берегу
374:   {
375:     switch (Koza_pos)
376:     {
377:       case 1: if ((Volk_pos==2)||(Kapusta_pos==2)) event=1; // Событие 1 - лодка занята
378:               else
379:               {
380:                 Koza_move(2);
381:                 if (Perev_pos==2) event=4; //Лодка готова!
382:               }
383:               break;
384:       case 2: Koza_move(1); break;
385:       default: event=3; // Коза на другом берегу!
386:     }
387:   } else // Лодка на правом берегу
388:   {
389:     switch (Koza_pos)
390:     {
```

```
391:       case 3: Koza_move(4); break;
392:       case 4: if ((Volk_pos==3)||(Kapusta_pos==3)) event=1; // Событие 1 - лодка занята
393:               else
394:               {
395:                 Koza_move(3);
396:                 if (Perev_pos==3) event=4; //Лодка готова!
397:               }
398:               break;
399:       default: event=3; // Коза на другом берегу!
400:     }
401:   }
402: }
403: //---------------------------------------------------------------------------
404: void KeyPressed_Kapusta(void) // Обработчик нажатий на кнопку Капусту
405: {
406:   if (Perev_pos<3) // Лодка на левом берегу
407:   {
408:     switch (Kapusta_pos)
409:     {
410:       case 1: if ((Koza_pos==2)||(Volk_pos==2)) event=1; // Событие 1 - лодка занята
411:               else
412:               {
413:                 Kapusta_move(2);
414:                 if (Perev_pos==2) event=4; //Лодка готова!
415:               }
416:               break;
417:       case 2: Kapusta_move(1); break;
418:       default: event=3; // Капуста на другом берегу!
419:     }
420:   } else // Лодка на правом берегу
421:   {
422:     switch (Kapusta_pos)
423:     {
424:       case 3: Kapusta_move(4); break;
425:       case 4: if ((Koza_pos==3)||(Volk_pos==3)) event=1; // Событие 1 - лодка занята
426:               else
427:               {
428:                 Kapusta_move(3);
429:                 if (Perev_pos==3) event=4; //Лодка готова!
430:               }
431:               break;
432:       default: event=3; // Капуста на другом берегу!
433:     }
434:   }
435: }
436: //---------------------------------------------------------------------------
437: void KeyPressed_HOD(void)
438: {
439:   uns8 Perev_new_pos, Volk_new_pos, Koza_new_pos, Kapusta_new_pos;
440:
441:   if ((Perev_pos==1)||(Perev_pos==4)) { event=2; return; }
442:   Volk_new_pos=Volk_pos;
443:   Koza_new_pos=Koza_pos;
444:   Kapusta_new_pos=Kapusta_pos;
445:   // Перевозчик находится в лодке, можно плыть:
446:   // Вычислим новые позиции персонажей и погасим тех, кто плывет:
447:   if (Perev_pos==2) // Едем с левого берега на правый
448:   {
449:     Perev_new_pos=3;
450:     if (Volk_pos==2) { Volk_new_pos=3; Volk_move(0);} // Если волк в лодке, то погасим его
451:     if (Koza_pos==2) { Koza_new_pos=3; Koza_move(0);} // Если коза в лодке, то погасим ее
452:     if (Kapusta_pos==2) { Kapusta_new_pos=3; Kapusta_move(0);} // Если капуста в лодке, то погасим
    ее
453:   } else // Едем с права на лево
454:   {
455:     Perev_new_pos=2;
456:     if (Volk_pos==3) { Volk_new_pos=2; Volk_move(0);} // Если волк в лодке, то погасим его
457:     if (Koza_pos==3) { Koza_new_pos=2; Koza_move(0);} // Если коза в лодке, то погасим ее
458:     if (Kapusta_pos==3) { Kapusta_new_pos=2; Kapusta_move(0);} // Если капуста в лодке, то погасим
    ее
459:   }
460:   Perev_move(0); // Перевозчик всегда плывет
461:   BlinkRiver(5); // Помигаем рекой
462:   // Зажигаем всех на (возможно) новых местах
463:   Perev_move(Perev_new_pos);
464:   Volk_move(Volk_new_pos);
465:   Koza_move(Koza_new_pos);
466:   Kapusta_move(Kapusta_new_pos);
```

```
467: }
468: //-------------------------------------------------------------------------------
469: void  IND_Check_Busy(void) // Ждать завершения операции индикатора
470: {
471:   DDRD = 0xFF;// Переключение шины данных на ввод
472:   PORTC = 0b110; // E=1 RW=1 RS=0
473:    while (BUSY_FLAG==1)
474:    {
475:      PORTC = 0b010; // Убрали E
476:      nop(); nop(); nop(); nop(); nop(); nop(); nop(); nop(); nop(); nop();
477:      PORTC = 0b110; // E=1
478:    }
479:    PORTC = 0;
480:    DDRD = 0;//Переключаем обратно на вывод шину данных
481: }
482: //-------------------------------------------------------------------------------
483: void IND_WriteIns(uns8 data)// Вывести инструкцию на индикатор
484: {
485:    IND_Check_Busy();//Ждем пока процессор занят
486:    PORTC = 0b100;//E=1 RW=0 RS=0
487:    PORTD = data;// Выводим байт на шину данных
488:    nop(); nop(); nop(); nop(); nop(); nop();
489:    PORTC = 0; // E RS убрали в 0
490: }
491: //-------------------------------------------------------------------------------
492: void IND_WriteData(uns8 data)// Вывести байт на индикатор
493: {
494:    IND_Check_Busy();//Ждем пока процессор занят
495:    PORTC = 0b101;//E=1 RW=0 RS=1
496:    PORTD = data;// Выводим байт на шину данных
497:    nop(); nop(); nop(); nop(); nop(); nop();
498:    PORTC = 0; // E RS убрали в 0
499: }
500: //-------------------------------------------------------------------------------
501: void IND_init(void) // Инициализация индикатора при включении
502: {
503:  delay_timer3_ms(100);
504:  IND_WriteIns(0x3A); // 0011 1010
505:  IND_WriteIns(0x0C); // Display ON/OFF: D=1 C=0 B=0
506:  IND_WriteIns(0x01); // Display clear
507:  IND_WriteIns(0x06); // Entry mode set
508: }
509:
510: //-------------------------------------------------------------------------------
511: char trans_char(char ch)
512: {
513:   switch (ch)
514:   {
515:     case 'А': return 0x41;
516:     case 'Б': return 0xA0;
517:     case 'В': return 0x42;
518:     case 'Г': return 0xA1;
519:     case 'Д': return 0xE0;
520:     case 'Е': return 0x45;
521:     case 'Ё': return 0x1B;
522:     case 'Ж': return 0xA3;
523:     case 'З': return 0xA4;
524:     case 'И': return 0xA5;
525:     case 'Й': return 0xA6;
526:     case 'К': return 0x4B;
527:     case 'Л': return 0xA7;
528:     case 'М': return 0x4D;
529:     case 'Н': return 0x48;
530:     case 'О': return 0x4F;
531:     case 'П': return 0xA8;
532:     case 'Р': return 0x50;
533:     case 'С': return 0x43;
534:     case 'Т': return 0x54;
535:     case 'У': return 0xA9;
536:     case 'Ф': return 0xAA;
537:     case 'Х': return 0x58;
538:     case 'Ц': return 0xE1;
539:     case 'Ч': return 0xAB;
540:     case 'Ш': return 0xAC;
541:     case 'Щ': return 0xE2;
542:     case 'Ъ': return 0xAD;
543:     case 'Ы': return 0xAE;
544:     case 'Ь': return 0xC4;
```

```
545:     case 'Э': return 0xAF;
546:     case 'Ю': return 0xB0;
547:     case 'Я': return 0xB1;
548:     case 'а': return 0x61;
549:     case 'б': return 0xB2;
550:     case 'в': return 0xB3;
551:     case 'г': return 0xB4;
552:     case 'д': return 0xE3;
553:     case 'е': return 0x65;
554:     case 'ё': return 0xB5;
555:     case 'ж': return 0xB6;
556:     case 'з': return 0xB7;
557:     case 'и': return 0xB8;
558:     case 'й': return 0xB9;
559:     case 'к': return 0xBA;
560:     case 'л': return 0xBB;
561:     case 'м': return 0xBC;
562:     case 'н': return 0xBD;
563:     case 'о': return 0x6F;
564:     case 'п': return 0xBE;
565:     case 'р': return 0x70;
566:     case 'с': return 0x63;
567:     case 'т': return 0xBF;
568:     case 'у': return 0x79;
569:     case 'ф': return 0xE4;
570:     case 'х': return 0x87;
571:     case 'ц': return 0xE5;
572:     case 'ч': return 0xC0;
573:     case 'ш': return 0xC1;
574:     case 'щ': return 0xE6;
575:     case 'ъ': return 0xC2;
576:     case 'ы': return 0xC3;
577:     case 'ь': return 0xC4;
578:     case 'э': return 0xC5;
579:     case 'ю': return 0xC6;
580:     case 'я': return 0xC7;
581:   }
582:   return ch;
583: }
584: //-------------------------------------------------------------------------------
585: void Display_clear(void)
586: {
587:   IND_WriteIns(1);
588: }
589: //-------------------------------------------------------------------------------
590: void print(const char *line1, const char *line2)
591: {
592:   uns8 i;
593:
594:   Display_clear();
595:   IND_WriteIns(LINE1);// Установим Адрес для строки 1
596:   i=0; while ((line1[i]!=0)&&(i<20)) IND_WriteData(trans_char(line1[i++]));
597:   IND_WriteIns(LINE2);// Установим Адрес для строки 2
598:   i=0; while ((line2[i]!=0)&&(i<20)) IND_WriteData(trans_char(line2[i++]));
599: }
600: //-------------------------------------------------------------------------------
601: void show_win(void)
602: {
603:   print("Ура!!!","Задание выполнено!");
604:   start_audio(1);
605:   BlinkGreen(10);
606:   print("Для новой игры жми","кнопку НОВАЯ ИГРА!");
607:   while(1); // Зависнем до нажатия кнопки сброса
608: }
609: //-------------------------------------------------------------------------------
610: void show_fail_volk_koza(void)
611: {
612:   uns8 Volk, Koza, i;
613:   print("Беда!", "Волк съел Козу:(");
614:   start_audio(2);
615:   Volk=Volk_pos;
616:   Koza=Koza_pos;
617:   for (i=0;i<10;i++)
618:   {
619:     Volk_move(0); Koza_move(0); // Погасили персонажей
620:     delay_timer3_ms(BLINK_MS);
621:     Volk_move(Volk); Koza_move(Koza); // Зажгли на прежнем месте
622:     delay_timer3_ms(BLINK_MS);
```

```
623:   }
624:   BlinkRed(5);
625:   print("Для новой игры жми","кнопку НОВАЯ ИГРА!");
626:   while(1); // Зависнем до нажатия кнопки сброса
627: }
628: //-------------------------------------------------------------------------------
629: void show_fail_koza_kapusta(void)
630: {
631:   uns8 Koza, Kapusta, i;
632:   print("Беда!", "Коза съела Капусту:(");
633:   start_audio(3);
634:   Koza=Koza_pos;
635:   Kapusta=Kapusta_pos;
636:   for (i=0;i<10;i++)
637:   {
638:     Koza_move(0); Kapusta_move(0);  // Погасили персонажей
639:     delay_timer3_ms(BLINK_MS);
640:     Koza_move(Koza); Kapusta_move(Kapusta); // Зажгли на прежнем месте
641:     delay_timer3_ms(BLINK_MS);
642:   }
643:   BlinkRed(5);
644:   print("Для новой игры жми","кнопку НОВАЯ ИГРА!");
645:   while(1); // Зависнем до нажатия кнопки сброса
646: }
647: //-------------------------------------------------------------------------------
648: void check_events (void)
649: {
650:   // Сначала проверим, нет ли ситуации выигрыша
651:   if ((Perev_pos==4)&&(Volk_pos==4)&&(Koza_pos==4)&&(Kapusta_pos==4)) show_win(); // Из этой
     функции нет выхода
652:   // Рассмотрим возможные варианты проигрыша
653:   if (Perev_pos>2) // Если перевозчик на правом берегу
654:   {
655:     if ((Volk_pos==1)&&(Koza_pos==1)) show_fail_volk_koza();
656:     if ((Koza_pos==1)&&(Kapusta_pos==1)) show_fail_koza_kapusta();
657:   } else // Если перевозчик на правом берегу
658:   {
659:     if ((Volk_pos==4)&&(Koza_pos==4)) show_fail_volk_koza();
660:     if ((Koza_pos==4)&&(Kapusta_pos==4)) show_fail_koza_kapusta();
661:   }
662:   // Теперь отработаем текущее событие (если оно имеется)
663:   switch (event)
664:   {
665:     case 0: Display_clear(); break;
666:     case 1: print("Ошибка! В лодке","только одно место!"); start_audio(4); break;
667:     case 2: print("Без Перевозчика","лодка не поплывет!"); start_audio(5); break;
668:     case 3: print("Лодка на другом","берегу!"); start_audio(6); break;
669:     case 4: print("Лодка готова!","Жми кнопку ПОПЛЫЛИ!"); break;
670:   }
671:   event=0;
672: }
673: //-------------------------------------------------------------------------------
674: void usart_txb (uns8 byte)
675: {
676:  while (TX2IF==0); // Пока нет флага прерывания загружать новое значение нельзя
677:  TXREG2=byte;
678: }
679: //-------------------------------------------------------------------------------
680: void check_usart (void) // Проверить, нет ли команды на загрузку ПЗУ от компьютера
681: {
682:   uns8 data;
683:   bit isFirstStart;
684:
685:   if (RC2IF==0) return;
686:   data=RCREG2; // При этом аппаратно сбрасывается запрос прерываний
687:   switch (kia_stat)
688:   {
689:     case 0: if (data=='L') kia_stat++; break;
690:     case 1: if (data=='O') kia_stat++; else kia_stat=0; break;
691:     case 2: if (data=='A') kia_stat++; else kia_stat=0; break;
692:     case 3: if (data=='D') kia_stat++; else kia_stat=0; break;
693:     case 4: // Здесь поступает размер буфера (младший байт):
694:             nbuf.low8=data; kia_stat++; break;
695:     case 5: // Здесь поступает размер буфера (средний байт):
696:             nbuf.mid8=data; kia_stat++; break;
697:     case 6: // Здесь поступает размер буфера (старший байт):
698:             nbuf.high8=data; kia_stat++;
699:             // Передадим размер как подтверждение
```

```
700:             usart_txb(nbuf.low8);
701:             usart_txb(nbuf.mid8);
702:             usart_txb(nbuf.high8);
703:             kia_count=0; // Обнулим счетчик принятых байтов
704:             kia_cs=0; // Контрольная сумма
705:             print("Начинаю обмен", "данными для ПЗУ");
706:             break;
707:     case 7: // Здесь приходит nbuf байт данных блоками по 256:
708:             if (kia_count.low8==0) // В начале каждой страницы
709:             {
710:               isFirstStart=1;
711:               do // Необходимо убежиться, что микросхема не занята записью предыдущего блока
712:               {
713:                 // Включаем формирование признака СТАРТ на шине:
714:                 if (isFirstStart) {SEN=1; isFirstStart=0;} else RSEN=1;
715:                 wait_ssp_complete();
716:                 // Передаем адрес микросхемы (управляющий байт)
717:                 if (kia_count<MIDN) SSPBUF=0b10100000; else SSPBUF=0b10100010;
718:                 wait_ssp_complete();
719:               } while (ACKSTAT==1); // Реально микросхема тратит 2,8 мс на запись блока
720:               usart_txb('Y'); // Подтвердим, что цикл ожидания завершился
721:               // Передаем старший байт - адрес ячейки
722:               SSPBUF=kia_count.mid8; wait_ssp_complete();
723:               // Передаем младший байт адреса ячейки
724:               SSPBUF=0; wait_ssp_complete();
725:             }
726:             // Передаем байт данных для ячейки
727:             SSPBUF=data; wait_ssp_complete();
728:             kia_cs+=data; // Подсчитаем контрольную сумму записанного блока
729:             kia_count++;  // Байт обработан, ставим указатель на следущий
730:             // Проверим, был ли это последний байт
731:             if ((kia_count.low8==0)||(kia_count==nbuf))
732:             {
733:               // Включаем форvирование признака СТОП на шине
734:               PEN=1; wait_ssp_complete();
735:             }
736:             if (kia_count==nbuf)
737:             {
738:               usart_txb(kia_cs);
739:               kia_stat=0;
740:               print("Запись в ПЗУ", "завершена");
741:             }
742:   }
743: }
744: //*****************************************************************************
745: void main(void)
746: {
747:   bit status, status_prev;
748:
749:   // Настраиваем порты:
750:   // 0 - выход, 1 - вход
751:   PORTA=0b10001100; // Отключим подтяжку порта Б, ключи отпущены
752:   sPORTB=0; PORTB=sPORTB; DDRB=1;
753:   sPORTC=0; PORTC=sPORTC; DDRC=0; // На шине управления индикатором все нули
754:   PORTD=0; DDRD=0;
755:   DDRE=0x0F; // Ввод 4-х кнопок
756:   sPORTF=0; PORTF=sPORTF; DDRF=0;
757:   sPORTG=0; PORTG=sPORTG; DDRG=0b01000000;
758:
759:   // Инициализируем переменные:
760:   kia_stat=0;
761:   event = 0;
762:   status_prev=0;
763:   idiv=0;
764:   isAudioStart=0;
765:   Perev_move(1);
766:   Volk_move(1);
767:   Koza_move(1);
768:   Kapusta_move(1);
769:   // Настраиваем ШИМ:
770:   // Настраиваем таймер 1
771:   PR1=200;// При значении 200 T=33,33 а F шим =30КГц
772:   TMR1IE=1;//Разрешим прерывания от таймера 1
773:   PEIE=1;// Разрешим прерывания от периферии
774:   TMR1ON=1;// Запустим таймер 1
775:   // Для управления шим будем использовать старшие 8 бит регистра длительности
776:   // Диапазон регулирования скважности от 0 до 200
777:   PW1DCH=100;// Нулевой уровень звукового сигнала
```

```
778:   PWM1ON=1;// Включили шим 1, меандр пошел
779:   GLINTD=0;// Разрешим глобальные прерывания
780:   //Требования к звуковым файлам:
781:   //Частота дискретизации 6КГц
782:   //Допустимый диапазон амплитуд от 0 до 200, нулевой уровень - 100
783:   // Максимальная длительность 21.8с
784:   // Включаем и настраиваем модуль I2C для работы с ЭППЗУ:
785:   // Выводы SCL и SDA уже установлены как входы автоматически при включении питания
786:   SSPCON1=0b00001000; // I2C ведущий, модуль пока выключен
787:   SSPADD=5; // При такте 24МГц и значении 5 получается частота шины 1МГц
788:   STAT_SMP=1; // Отключить контроль фронтов
789:   SSPEN=1; // Включить модуль СПП в режиме I2C
790:   //НАстраиваем USART-2 на работу в режиме 38400 8-N-1
791:   SPBRG2=9; // Уст. скорость 38400
792:   SPEN2=1; // Включаем USART-2, выводы RX, TX настроятся сами, остальные по умолчанию
793:   CREN2=1; // Включаем приемник
794:   TXEN2=1; // Включаем передатчик
795:   //----------------------------
796:   IND_init(); // Настраиваем индикатор
797:   print("Помоги перевозчику!", "Нажимай кнопки!"); start_audio(0);
798:   while (1)
799:   {
800:     status=KN_PEREV||KN_VOLK||KN_KOZA||KN_KAPUSTA||!KN_HOD;
801:     if ((status==1)&&(status_prev==0))
802:     {
803:       if (KN_PEREV) KeyPressed_Perev();
804:       else if (KN_VOLK) KeyPressed_Volk();
805:       else if (KN_KOZA) KeyPressed_Koza();
806:       else if (KN_KAPUSTA) KeyPressed_Kapusta();
807:       else KeyPressed_HOD();
808:       check_events(); // После нажатия на любую клавишу обработаем события
809:     }
810:     status_prev=status;
811:     // Проверим не пришел ли байт запроса от USART
812:     check_usart();
813:   }
814: }
815: //*****************************************************************************
816:
```