Университет ИТМО
ПИиКТ

Лабораторная работа №2: «Последовательный интерфейс UART»

По дисциплине: Проектирование вычислительных систем

Вариант: 3

Выполнили:
Неграш А. В., P34301
Перевозчиков И. С., P34301

Преподаватель:
Пинкевич В. Ю.

Санкт-Петербург
2023

1. Задание

Разработать и реализовать два варианта драйверов UART для стенда SDK-1.1M: с использованием и без использования прерываний. Драйверы, использующие прерывания, должны обеспечивать работу в «неблокирующем» режиме (возврат из функции происходит сразу же, без ожидания окончания приема/отправки), а также буферизацию данных для исключения случайной потери данных. В драйвере, не использующем прерывания, функция приема данных также должна быть «неблокирующей», то есть она не должна зависать до приема данных (которые могут никогда не поступить). При использовании режима «без прерываний» прерывания от соответствующего блока UART должны быть запрещены.

Написать с использованием разработанных драйверов программу, которая выполняет определенную вариантом задачу. Для всех вариантов должно быть реализовано два режима работы программы: с использованием и без использования прерываний. Каждый принимаемый стендом символ должен отсылаться обратно, чтобы он был выведен в консоли (так называемое «эхо»). Каждое новое сообщение от стенда должно выводиться с новой строки. Если вариант предусматривает работу с командами, то на каждую команду должен выводиться ответ, определенный в задании или «ОК», если ответ не требуется. Если введена команда, которая не поддерживается, должно быть выведено сообщение об этом.

Вариант 3:

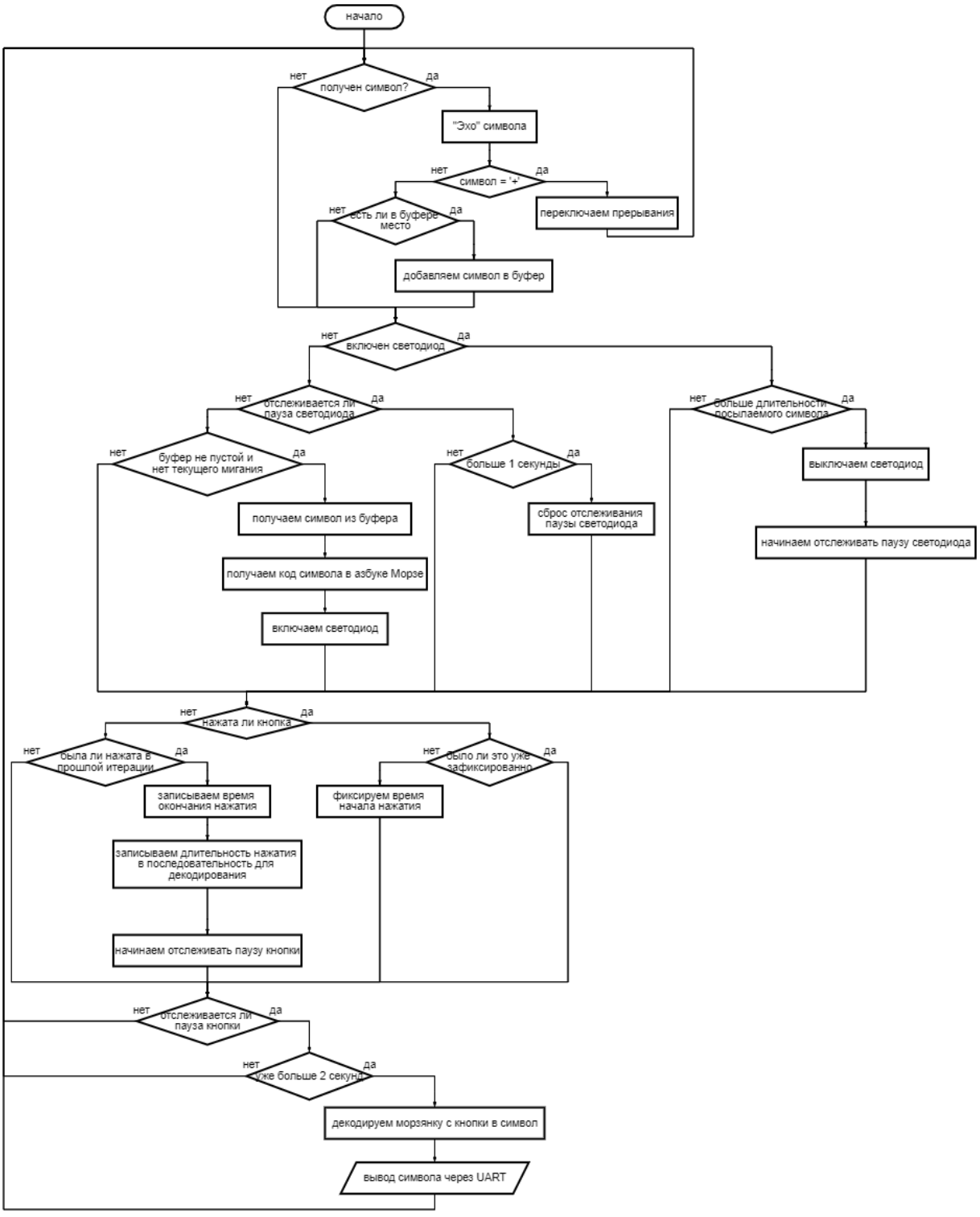
Доработать программу, посылающую сигналы азбуки Морзе для латинского алфавита. Последовательность точек и тире, полученных нажатием кнопки стенда, должна дешифроваться и отправляться через последовательный канал в виде символов (букв латинского алфавита). Символы, получаемые стендом через последовательный канал, должны шифроваться и «проигрываться» с помощью светодиода. Если в момент «мигания» приходят новые символы, они добавляются в очередь. Должна быть возможность одновременной буферизации до восьми символов: они должны запоминаться стендом и последовательно проигрываться.

Считывание нажатий кнопки и отправка дешифрованных символов должна функционировать одновременно с приёмом через последовательный канал и миганием светодиода.

Включение/выключение прерываний должно осуществляться при получении стендом символа «+».

Скорость обмена данными по UART: 115200 бит/с.

2. Блок-схема прикладного алгоритма



3. Исходный код

3.1. Файл usart.c

```
int uart6_recieve_finished;
int uart6_transmit_ongoing;

static uint8_t uart6_buf;

HAL_StatusTypeDef uart6_start_recieve_char_it() {
	uart6_recieve_finished = 0;
	return HAL_UART_Receive_IT(&huart6, &uart6_buf, 1);
}

int uart6_try_get_received_char(uint8_t *buf) {
	if (uart6_recieve_finished) {
		*buf = uart6_buf;
		return 1;
	}

	return 0;
}

HAL_StatusTypeDef uart6_transmit_it (uint8_t *buf, int len) {
	while (uart6_transmit_ongoing);

	uart6_transmit_ongoing = 1;
	return HAL_UART_Transmit_IT(&huart6, buf, len);
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
	if (huart->Instance == USART6) {
		uart6_recieve_finished = 1;
	}
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
	if (huart->Instance == USART6) {
		uart6_transmit_ongoing = 0;
	}
}
```

3.2. Файл main.c

```
char letters[] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
int morse[] = {22222, 12222, 11222, 11122, 11112, 11111, 21111, 22111, 22211, 22221, 12,
2111, 2121, 211, 1, 1121, 221, 1111, 11, 1222, 212, 1211, 22, 21, 222, 1221, 2212, 121, 111, 2,
112, 1112, 122, 2112, 2122, 2211};
_Bool isUseInterrupts = false;
char buffer[8];
int bufferGetPosition = 0;
int bufferAddPosition = 0;

_Bool isPressed() { return HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_15) == GPIO_PIN_RESET; }

_Bool isNotPressed() { return HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_15) == GPIO_PIN_SET; }

_Bool isBufferEmpty() { return (buffer[0] == 0 && buffer[1] == 0 && buffer[2] == 0 && buffer[3]
== 0 && buffer[4] == 0 && buffer[5] == 0 && buffer[6] == 0 && buffer[7] == 0); }

_Bool isBufferFull() { return (buffer[0] != 0 && buffer[1] != 0  && buffer[2] != 0 && buffer[3] != 0
&& buffer[4] != 0 && buffer[5] != 0 && buffer[6] != 0 && buffer[7] != 0); }

void turnOn() { HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET); }

void turnOff() { HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET); }

void addToBuffer(char s) {
	buffer[bufferAddPosition] = s;
	bufferAddPosition++;
	if (bufferAddPosition > 7) {
		bufferAddPosition = 0;
	}
}

char getCharFromBuffer() {
	char toReturn = buffer[bufferGetPosition];
	buffer[bufferGetPosition] = 0;
	bufferGetPosition++;
	if (bufferGetPosition > 7) {
		bufferGetPosition = 0;
	}
	return toReturn;
}

int getMorseCode(char s) {
	int position = -1;
	for (int i = 0; i < sizeof(letters); i++) {
		if (s == letters[i]) {
			position = i;
			break;
		}
	}
```

```
        if (position >= 0) {
                return morse[position];
        } else {
                return position;
        }
}

char getCharByMorseCode(int s) {
        int position = -1;
        for (int i = 0; i < sizeof(morse); i++) {
                if (s == morse[i]) {
                        position = i;
                        break;
                }
        }
        if (position >= 0) {
                return letters[position];
        } else {
                return '-';
        }
}

<...>

   char recieved, current, toSend[2];
   int currentMorse = -1;
   int morseTen = 5;
   int btnMorse = 0;
   int btnCounter = 0;
   _Bool isLightOn = false;
   _Bool isPause = false;
   _Bool isBtnPressed = false;
   _Bool isBtnPause = false;
   uint32_t blinkStartTime, pauseStartTime, diffLO, diffPause, btnStart, btnDiff, btnPauseStart,
btnPauseDiff;
   int blinkingMode;
   char err[] = "Unsupported symbol by button \n";
   char intOn[] = "Interrupts turned On\n";
   char intOff[] = "Interrupts turned Off\n";

   while (1) {
      if (!isUseInterrupts) {
         //No Interrupts
         if (HAL_OK == HAL_UART_Receive(&huart6, (uint8_t *) &recieved, 1, 1)) {
            char toret[2];
            toret[0] = recieved;
            toret[1] = '\n';
```

```
    HAL_UART_Transmit(&huart6, (uint8_t *) toret, 2, 10);
    if (recieved == '+') {
      isUseInterrupts = true;
      HAL_UART_Transmit(&huart6, (uint8_t *) intOn, sizeof(intOn), 10);
      HAL_NVIC_EnableIRQ(USART6_IRQn);
      uart6_start_recieve_char_it();
      continue;
    }
    if (!isBufferFull()) {
       addToBuffer(recieved);
    }
  }

  if (!isLightOn && !isPause) {
    if (!isBufferEmpty() && currentMorse < 0) {
      current = getCharFromBuffer();
      currentMorse = getMorseCode(current);
    }

    if (currentMorse >= 0) {
      blinkingMode = currentMorse / pow(10, morseTen);
      currentMorse = currentMorse % (int)pow(10, morseTen);
      morseTen--;
      if (morseTen == -1) {
        morseTen = 5;
        currentMorse = -1;
      }

      if (blinkingMode > 0) {
        blinkStartTime = HAL_GetTick();
        turnOn();
        isLightOn = true;
      }
    }
  }

  if (isLightOn) {
    diffLO = HAL_GetTick() - blinkStartTime;
    if (blinkingMode == 1) {
      if (diffLO >= 300) {
        turnOff();
        pauseStartTime = HAL_GetTick();
        isLightOn = false;
        isPause = true;
      }
    } else if (blinkingMode == 2) {
      if (diffLO >= 700) {
```

```
            turnOff();
            pauseStartTime = HAL_GetTick();
            isLightOn = false;
            isPause = true;
        }
    }
}

if (isPause) {
    diffPause = HAL_GetTick() - pauseStartTime;
    if (diffPause >= 300) {
        isPause = false;
    }
}

if (!isBtnPressed) {
    if (isPressed()) {
        isBtnPressed = true;
        isBtnPause = false;
        btnStart = HAL_GetTick();
    }
}

if (isBtnPressed) {
    if (isNotPressed()) {
        isBtnPressed = false;
        isBtnPause = true;
        btnDiff = HAL_GetTick() - btnStart;
        btnPauseStart = HAL_GetTick();
        if (btnDiff >= 100) {
            if (btnDiff < 500) {
                btnMorse = btnMorse * 10 + 1;
                btnCounter++;
            } else {
                btnMorse = btnMorse * 10 + 2;
                btnCounter++;
            }
        }
    }
}

if (isBtnPause) {
    if (isNotPressed()) {
        btnPauseDiff = HAL_GetTick() – btnPauseStart;
        if (btnPauseDiff > 1000) {
            toSend[0] = getCharByMorseCode(btnMorse);
            if (toSend[0] == '-') {
```

```
                HAL_UART_Transmit(&huart6, (uint8_t *) err, sizeof(err), 10);
            } else {
                toSend[1] = '\n';
                HAL_UART_Transmit(&huart6, (uint8_t *) toSend, 2, 10);
            }
            isBtnPause = false;
            btnMorse = 0;
            btnCounter = 0;
        }
    }
  }
} else {
//With Interrupts
if (uart6_try_get_received_char((uint8_t *) &recieved)) {
  uart6_start_recieve_char_it();
  char toret[2];
  toret[0] = recieved;
  toret[1] = '\n';
  uart6_transmit_it((uint8_t *) &toret, sizeof(toret));
  if (recieved == '+') {
    isUseInterrupts = false;
    HAL_UART_Abort_IT(&huart6);
    HAL_NVIC_DisableIRQ(USART6_IRQn);
    HAL_UART_Transmit(&huart6, (uint8_t *) intOff, sizeof(intOff), 10);
    continue;
  }

  if (!isBufferFull()) {
    addToBuffer(recieved);
  }
}

if (!isLightOn && !isPause) {
  if (!isBufferEmpty() && currentMorse < 0) {
    current = getCharFromBuffer();
    currentMorse = getMorseCode(current);
  }

  if (currentMorse >= 0) {
    blinkingMode = currentMorse / pow(10, morseTen);
    currentMorse = currentMorse % (int)pow(10, morseTen);
    morseTen--;
    if (morseTen == -1) {
      morseTen = 5;
      currentMorse = -1;
    }
```

```
      if (blinkingMode > 0) {
        blinkStartTime = HAL_GetTick();
        turnOn();
        isLightOn = true;
      }
    }
}

if (isLightOn) {
    diffLO = HAL_GetTick() - blinkStartTime;
    if (blinkingMode == 1) {
      if (diffLO >= 300) {
        turnOff();
        pauseStartTime = HAL_GetTick();
        isLightOn = false;
        isPause = true;
      }
    } else if (blinkingMode == 2) {
      if (diffLO >= 700) {
        turnOff();
        pauseStartTime = HAL_GetTick();
        isLightOn = false;
        isPause = true;
      }
    }
}

if (isPause) {
    diffPause = HAL_GetTick() - pauseStartTime;
    if (diffPause >= 300) {
      isPause = false;
    }
}

if (!isBtnPressed) {
    if (isPressed()) {
      isBtnPressed = true;
      isBtnPause = false;
      btnStart = HAL_GetTick();
    }
}

if (isBtnPressed) {
    if (isNotPressed()) {
      isBtnPressed = false;
      isBtnPause = true;
      btnDiff = HAL_GetTick() - btnStart;
```

```
            btnPauseStart = HAL_GetTick();
            if (btnDiff >= 100) {
              if (btnDiff < 500) {
                btnMorse = btnMorse * 10 + 1;
                btnCounter++;
              } else {
                btnMorse = btnMorse * 10 + 2;
                btnCounter++;
              }
            }
          }
        }

        if (isBtnPause) {
          if (isNotPressed()) {
            btnPauseDiff = HAL_GetTick() - btnPauseStart;
            if (btnPauseDiff > 1000) {
              toSend[0] = getCharByMorseCode(btnMorse);
              if (toSend[0] == '-') {
                uart6_transmit_it((uint8_t *) &err, sizeof(err));
              } else {
                toSend[1] = '\n';
                uart6_transmit_it((uint8_t *) &toSend, sizeof(toSend));
              }
              isBtnPause = false;
              btnMorse = 0;
              btnCounter = 0;
            }
          }
        }
      }
    }
```

4. Вывод

Итак, в процессе выполнения данной лабораторной работы мы продолжили ознакомление с устройством стенда SDK 1.1M и изучили работу с интерфейсом UART, а также разработали собственную программу для вывода и считывания символов азбуки Морзе, в которой продемонстрировали полученные знания.