

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра микроэлектроники, информационных технологий и управляющих
систем (МИТиУС)

GPIO, MORZE

Отчет по лабораторной работе №2

по дисциплине «Аппаратные Средства Телекоммуникационных Систем»

Студенты гр. 735:

_____ Е. Ю. Борисова

_____ И. В. Забатурина

_____ Д. А. Осипов

Принял:

Ст. преподаватель кафедры
МИТУС

_____ С. П. Недяк
(оценка)

Инженер кафедры МИТУС

_____ Ю. Б. Шаропин
(оценка)

(дата)

1 Введение

Цель лабораторной работы: изучить основы приёма и передачи пользовательской информации посредством интерфейса GPIO (портов ввода-вывода микроконтроллера) и программного управления этим процессом.

Задача: реализовать кодирование информации с помощью азбуки Морзе и организовать сеть из микроконтроллеров, обменивающийся информацией по протоколу с кодом Морзе; в узлах этой сети может находиться как человек, так и микроконтроллер (МК).

2 Теория

Морзе. Азбука Морзе – это способ знакового кодирования, представление букв алфавита и цифр последовательностью символов: длинных тире и коротких точек.

Порты ввода-вывода

В Reference manual на STM32 приведена схема порта ввода-вывода (рисунок 2.3) более подробная по сравнению с спецификацией на микроконтроллер фирмы Миландр. На данной схеме приведена комплиментарная пара транзисторов в блоке вывода. (Рисунок 2.1 – 2.3)

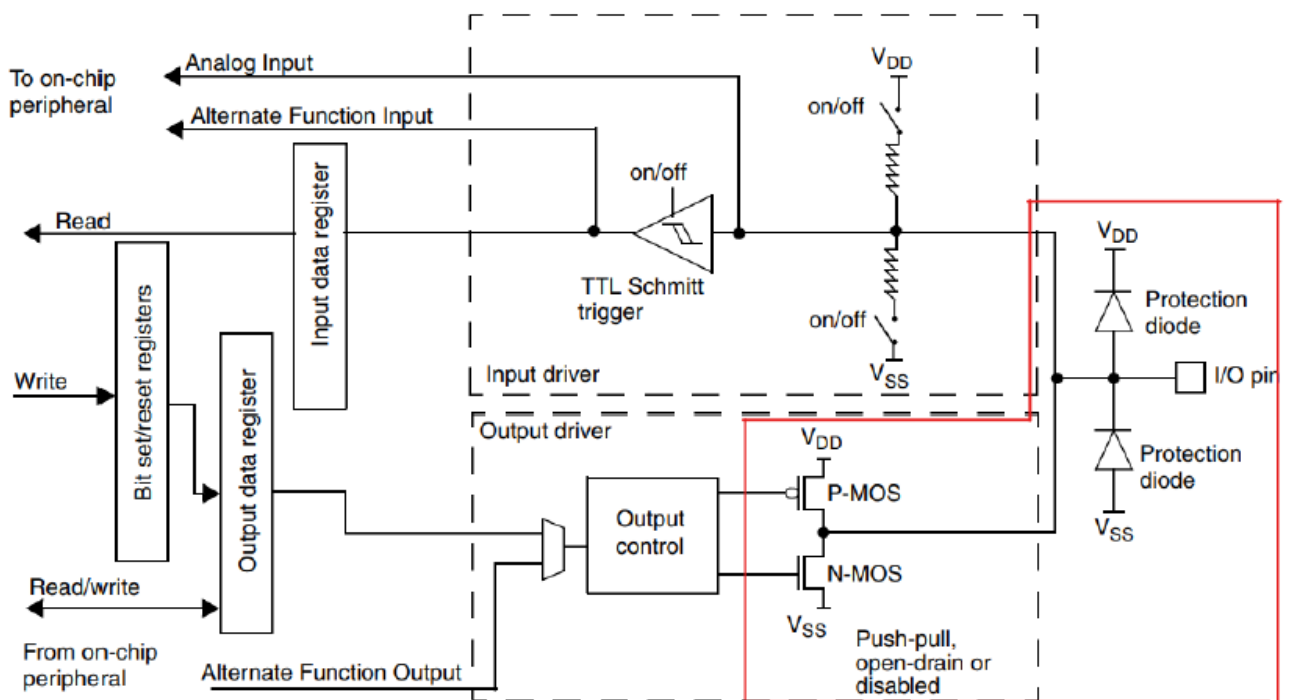


Рисунок 2.1 – Схема порта ввода-вывода

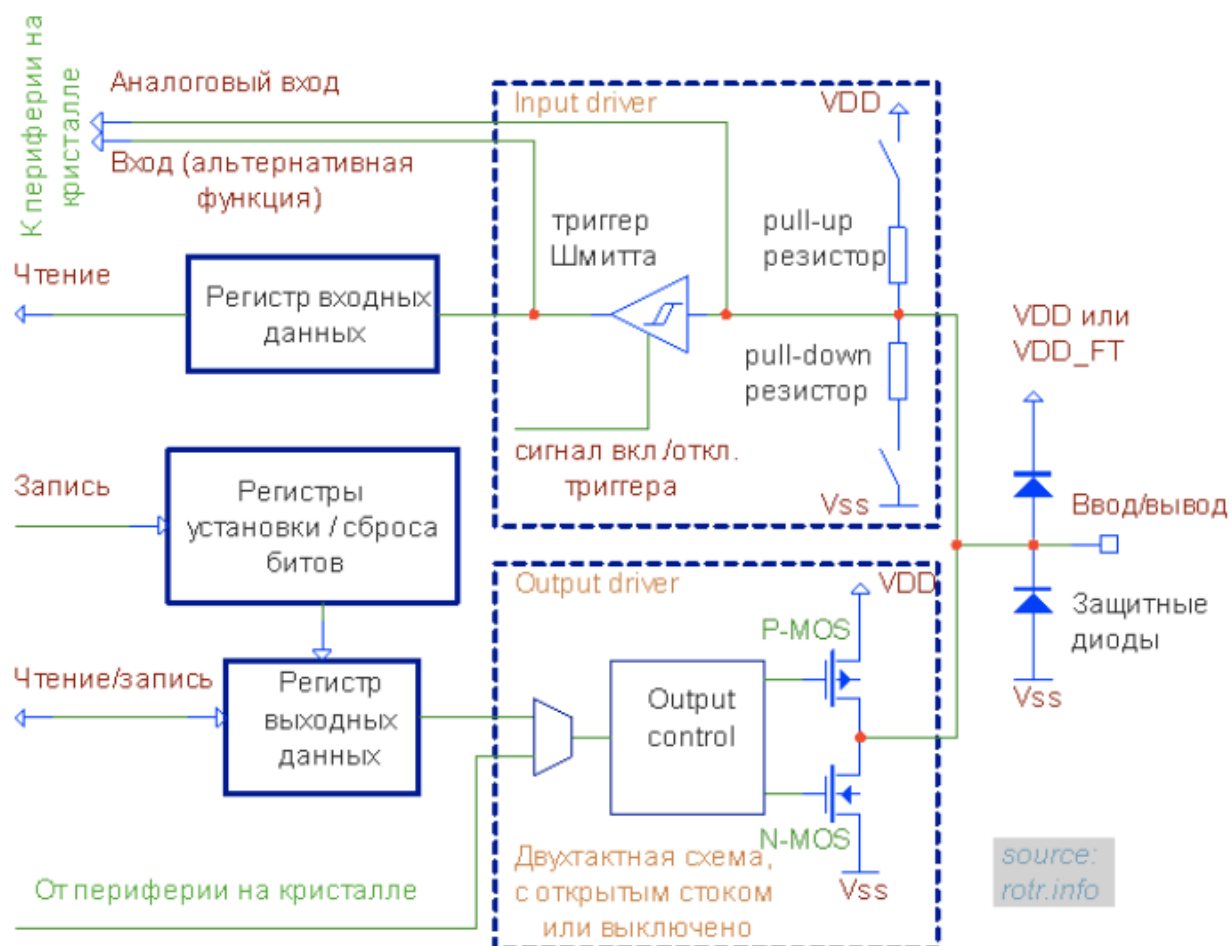


Рисунок 2.2 – Схема порта ввода-вывода на русском (спасибо А. С. Федурину)

Известно, что при работе порта в качестве вывода возможны два режима:

- Коротко о режимах. В режиме open-drain используется только один транзистор N-теоретически может быть только два возможных состояния:

- В режиме push-pull используются оба транзистора P-MOS и N-MOS, и теоретически может быть 4 возможных состояния:

- Логической единице на выводе соответствует состояние 3, логическому нулю - состояние 4.

Теоретически, в момент переключения вывода (из логической единицы в ноль или наоборот - не имеет значения) может произойти так, что оба транзистора открыты. Таким образом Vdd и Vss будут соединены напрямую (можно провести прямую линию между ними на схеме), сопротивления транзисторов малы, а потому по закону Ома протекающий ток будет очень велик. То есть произойдет короткое замыкание, от которого транзисторы сгорят, и чтобы этого не произошло, на аппаратном уровне существует блок управления Output Control, который не допустит, чтобы в момент переключения оба транзистора оказались открыты. Но при этом, фактически, в некоторый момент времени оба транзистора будут закрыты и ток не будет течь. Таким образом, в режиме push-pull используются состояния 1, 3 и 4, при этом состояние 1 является переходным.

Для работы с портами ввода/вывода используется библиотека MDR32F9Qx_port.h, которая описывает следующие регистры (рисунок 2.4):

- MDR_PORTA
- MDR_PORTB
- MDR_PORTC
- MDR_PORTD
- MDR_PORTE
- MDR_PORTF

Для инициализации используется структура типа PORT_InitTypeDef с полями:

- PORT_OE – направление передачи данных
- PORT_FUNC – режим работы вывода порта
- PORT_MODE – режим работы контроллера
- PORT_SPEED – скорость фронта вывода
- PORT_Pin – выбор выводов для инициализации

Адрес	Размер	Блок		Примечание
0x400A_8000	32 байт	21	PORTA	Регистры управления порта А
0x400B_0000	32 байт	22	PORTB	Регистры управления порта В
0x400B_8000	32 байт	23	PORTC	Регистры управления порта С
0x400C_0000	32 байт	24	PORTD	Регистры управления порта D
0x400C_8000	32 байт	25	PORTE	Регистры управления порта E

Рисунок 2.4 – Регистры портов ввода/вывода

Микроконтроллер имеет 6 портов ввода/вывода. Порты 16-разрядные, и их выходы мультиплексируются между различными функциональными блоками, управление для

каждого вывода отдельное. Для того, чтобы выводы порта перешли под управление того или иного периферийного блока, необходимо задать для нужных выводов выполняемую функцию и настройки.

В начале необходимо включить тактирование используемых портов (т.к на регистры портов должна поступить тактовая частота, иначе проект не будет работать)

RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);

Затем необходимо настроить порты ввода-вывода(см. спецификация1986BE9X.pdf, стр.177,таблица 120). (Рисунок 2.5)

Вывод	Аналоговая функция ANALOG_EN=0	Цифровая функция			
		Порт IO MODE[1:0]=00 ANALOG_EN=1	Основная MODE[1:0]=01 ANALOG_EN=1	Альтернативная MODE[1:0]=10 ANALOG_EN=1	Переопределенная MODE[1:0]=11 ANALOG_EN=1
Порт A					
PA0	-	PA0	DATA0	¹⁾ EXT_INT1	⁹⁾ -
PA1	-	PA1	DATA1	TMR1_CH1	TMR2_CH1
PA2	-	PA2	DATA2	TMR1_CH1N	TMR2_CH1N
PA3	-	PA3	DATA3	TMR1_CH2	TMR2_CH2
PA4	-	PA4	DATA4	TMR1_CH2N	TMR2_CH2N
PA5	-	PA5	DATA5	TMR1_CH3	TMR2_CH3
PA6	-	PA6	DATA6	CAN1_TX	²⁾ UART1_RXD
PA7	-	PA7	DATA7	CAN1_RX	UART1_TXD
PA8	-	PA8	DATA8	TMR1_CH3N	TMR2_CH3N
PA9	-	PA9	DATA9	TMR1_CH4	TMR2_CH4
PA10	-	PA10	DATA10	nUART1DTR	¹⁰⁾ TMR2_CH4N
PA11	-	PA11	DATA11	nUART1RTS	TMR2_BLK
PA12	-	PA12	DATA12	nUART1RI	TMR2_ETR
PA13	-	PA13	DATA13	nUART1DCD	TMR1_CH4N
PA14	-	PA14	DATA14	nUART1DSR	TMR1_BLK
PA15	-	PA15	DATA15	nUART1CTS	TMR1_ETR
Порт B					
PB0	-	PB0 JA_TDO	DATA16	¹⁾ TMR3_CH1	UART1_TXD
PB1	-	PB1 JA_TMS	DATA17	TMR3_CH1N	UART2_RXD
PB2	-	PB2 JA_TCK	DATA18	TMR3_CH2	CAN1_TX
PB3	-	PB3 JA_TDI	DATA19	TMR3_CH2N	CAN1_RX
PB4	-	PB4 JA_TRST	DATA20	TMR3_BLK	TMR3_ETR
PB5	-	PB5	DATA21	UART1_TXD	¹⁰⁾ TMR3_CH3
PB6	-	PB6	DATA22	UART1_RXD	TMR3_CH3N
PB7	-	PB7	DATA23	nSIROUT1	TMR3_CH4
PB8	-	PB8	DATA24	COMP_OUT	⁷⁾ TMR3_CH4N
PB9	-	PB9	DATA25	nSIRIN1	¹⁰⁾ EXT_INT4
PB10	-	PB10	DATA26	EXT_INT2	⁹⁾ nSIROUT1
PB11	-	PB11	DATA27	EXT_INT1	COMP_OUT
PB12	-	PB12	DATA28	SSP1_FSS	SSP2_FSS
PB13	-	PB13	DATA29	SSP1_CLK	SSP2_CLK

Рисунок 2.5 - Функции портов

Исходя из таблицы, мы видим, что у портов микроконтроллера есть аналоговая и цифровая функция.

Аналоговая отвечает за блоки АЦП, ЦАП.

Цифровая функция порта разделена на несколько видов. Основная, альтернативная и переопределенная отвечают за взаимодействие внутренних периферийных компонентов с выводами МК.

Для данной лабораторной работы необходима колонка таблицы, которая отвечает за использование портов как «Порт ИО».

Микроконтроллер имеет 6 портов ввода/вывода. Порты 16-разрядные и их выводы мультиплексируются между различными функциональными блоками, управление для каждого вывода отдельное. Для того, чтобы выводы порта перешли под управление того или иного периферийного блока, необходимо задать для нужных выводов выполняемую функцию и настройки. (Таблица 2.1)

Таблица 2.1 - Описание регистров портов ввода-вывода

Название	Описание	
MDR_PORTA	Порт А	
MDR_PORTB	Порт В	
MDR_PORTC	Порт С	
MDR_PORTD	Порт D	
MDR_PORTE	Порт E	
MDR_PORTF	Порт F	
RXTX[15:0]	MDR_PORTx->RXTX	Данные порта
OE[15:0]	MDR_PORTx->OE	Направление порта
FUNC[31:0]	MDR_PORTx->FUNC	Режим работы порта
ANALOG[15:0]	MDR_PORTx->ANALOG	Аналоговый режим работы порта
PULL[31:0]	MDR_PORTx->PULL	Подтяжка порта
PD[31:0]	MDR_PORTx->PD	Режим работы выходного драйвера
PWR[31:0]	MDR_PORTx->PWR	Режим мощности передатчика
GFEN[31:0]	MDR_PORTx->GFEN	Режим работы входного фильтра

PORT_Pin – выбор выводов для инициализации

Регистр OE - определяет режим работы порта (направление передачи данных):

- ☐ ввод PORT_OE_IN (0);
- ☐ вывод PORT_OE_OUT (1)

Регистр FUNC - определяет режим работы вывода порта:

- ☐ Порт PORT_FUNC_PORT (0);
- ☐ Основная функция PORT_FUNC_MAIN (1);
- ☐ Альтернативная функция PORT_FUNC_ALTER (2);
- ☐ Переопределенная функция PORT_FUNC_OVERRIDE (3)

Регистр MODE - определяет режим работы контроллера:

- ☐ аналоговый PORT_MODE_ANALOG(0);
- ☐ цифровой PORT_MODE_DIGITAL(1)

Регистр SPEED- определяет скорость работы порта:

- ☐ зарезервировано (передатчик отключен) PORT_OUTPUT_OFF (0);
- ☐ медленный фронт (порядка 100 нс) PORT_SPEED_SLOW (1);
- ☐ быстрый фронт (порядка 20 нс) PORT_SPEED_FAST (2);
- ☐ максимально быстрый фронт (порядка 10 нс)

Сначала выбираем режим работы с портом к которому хотим обратиться. Необходимо перед выполнением функции на порте поменять значение MODE в одно из значений представленных в таблице для порта с которым будем работать.

Аналоговая функция ANALOG_EN=0	Порт IO MODE[1:0]=00 ANALOG_EN=1	Основная MODE[1:0]=01 ANALOG_EN=1	Альтернативная MODE[1:0]=10 ANALOG_EN=1	Переопределенная MODE[1:0]=11 ANALOG_EN=1
--------------------------------------	--	---	---	---

Рисунок 2.6 – Режимы работы портов

Допустим сделать подтяжку напряжения на порту PA1 и хотим обратиться к цифровому порту TMR1_CH1. В таблице – «Порты ввода-вывода». (Рисунок 2.7 – 2.999)

Значение MODE должно быть равно «10» и ANALOG_EN=1.

Вывод	Аналоговая функция ANALOG_EN=0	Цифровая функция					
		Порт IO MODE[1:0]=00 ANALOG_EN=1	Основная MODE[1:0]=01 ANALOG_EN=1	Альтернативная MODE[1:0]=10 ANALOG_EN=1	Переопределенная MODE[1:0]=11 ANALOG_EN=1		
Порт A							
PA0	-	PA0	DATA0	1)	EXT INT1	9)	-
PA1	-	PA1	DATA1		TMR1_CH1		TMR2_CH1
PA2	-	PA2	DATA2		TMR1_CH1N		TMR2_CH1N
PA3	-	PA3	DATA3		TMR1_CH2		TMR2_CH2
PA4	-	PA4	DATA4		TMR1_CH2N		TMR2_CH2N
PA5	-	PA5	DATA5		TMR1_CH3		TMR2_CH3
PA6	-	PA6	DATA6		CAN1_TX	2)	UART1_RXD
PA7	-	PA7	DATA7		CAN1_RX		UART1_TXD
PA8	-	PA8	DATA8		TMR1_CH3N		TMR2_CH3N
PA9	-	PA9	DATA9		TMR1_CH4		TMR2_CH4
PA10	-	PA10	DATA10		nUART1DTR	10)	TMR2_CH4N
PA11	-	PA11	DATA11		nUART1RTS		TMR2_BLK
PA12	-	PA12	DATA12		nUART1RI		TMR2_ETR
PA13	-	PA13	DATA13		nUART1DCD		TMR1_CH4N
PA14	-	PA14	DATA14		nUART1DSR		TMR1_BLK
PA15	-	PA15	DATA15		nUART1CTS		TMR1_ETR

Рисунок 2.7 - Порты ввода-вывода – 1 часть

Порт В							
PB0	-	PB0	JA_TDO	DATA16	1)	TMR3_CH1	UART1_TXD
PB1	-	PB1	JA_TMS	DATA17		TMR3_CH1N	UART2_RXD
PB2	-	PB2	JA_TCK	DATA18		TMR3_CH2	CAN1_TX
PB3	-	PB3	JA_TDI	DATA19		TMR3_CH2N	CAN1_RX
PB4	-	PB4	JA_TRST	DATA20		TMR3_BLK	TMR3_ETR
PB5	-	PB5		DATA21		UART1_TXD	10) TMR3_CH3
PB6	-	PB6		DATA22		UART1_RXD	TMR3_CH3N
PB7	-	PB7		DATA23		nSIROUT1	TMR3_CH4
PB8	-	PB8		DATA24		COMP_OUT	7) TMR3_CH4N
PB9	-	PB9		DATA25		nSIRIN1	10) EXT_INT4
PB10	-	PB10		DATA26		EXT_INT2	9) nSIROUT1
PB11	-	PB11		DATA27		EXT_INT1	COMP_OUT
PB12	-	PB12		DATA28		SSP1_FSS	SSP2_FSS
PB13	-	PB13		DATA29		SSP1_CLK	SSP2_CLK
PB14	-	PB14		DATA30		SSP1_RXD	SSP2_RXD
PB15	-	PB15		DATA31		SSP1_TXD	SSP2_TXD

Рисунок 2.8 - Порты ввода-вывода – 2 часть

Порт С							
PC0	-	PC0	READY ¹⁷⁾	1)	SCL1	11)	SSP2_FSS
PC1	-	PC1	OE		SDA1		SSP2_CLK
PC2	-	PC2	WE		TMR3_CH1	12)	SSP2_RXD
PC3	-	PC3	BE0		TMR3_CH1N		SSP2_TXD
PC4	-	PC4	BE1		TMR3_CH2		TMR1_CH1
PC5	-	PC5	BE2		TMR3_CH2N		TMR1_CH1N
PC6	-	PC6	BE3		TMR3_CH3		TMR1_CH2
PC7	-	PC7	CLOCK		TMR3_CH3N		TMR1_CH2N
PC8	-	PC8	CAN1_TX	2)	TMR3_CH4		TMR1_CH3
PC9	-	PC9	CAN1_RX		TMR3_CH4N		TMR1_CH3N
PC10	-	PC10	-		TMR3_ETR		TMR1_CH4
PC11	-	PC11	-		TMR3_BLK		TMR1_CH4N
PC12	-	PC12	-		EXT_INT2		TMR1_ETR
PC13	-	PC13	-		EXT_INT4	9)	TMR1_BLK
PC14	-	PC14	-		SSP2_FSS	13)	CAN2_RX
PC15	-	PC15	-		SSP2_RXD		CAN2_TX

Рисунок 2.9 - Порты ввода-вывода – 3 часть

Порт D									
PD0	ADC0 REF+	5	PD0 JB TMS	TMR1 CH1N	3)	UART2 RXD	14)	TMR3 CH1	
PD1	ADC1 REF-		PD1 JB TCK	TMR1 CH1		UART2 TXD		TMR3 CH1N	
PD2	ADC2		PD2 JB TRST	BUSY1	1)	SSP2 RXD	13)	TMR3 CH2	
PD3	ADC3		PD3 JB TDI	-		SSP2 FSS		TMR3 CH2N	
PD4	ADC4		PD4 JB TDO	TMR1 ETR		nSIROUT2	14)	TMR3 BLK	
PD5	ADC5		PD5	CLE	1)	SSP2 CLK	13)	TMR2 ETR	
PD6	ADC6		PD6	ALE		SSP2 TXD		13)	TMR2 BLK
PD7	ADC7		PD7	TMR1 BLK	3)	nSIRIN2	14)	UART1 RXD	
PD8	ADC8		PD8	TMR1 CH4N		TMR2 CH1		UART1 TXD	
PD9	ADC9		PD9	CAN2 TX	4)	TMR2 CH1N		SSP1 FSS	
PD10	ADC10		PD10	TMR1 CH2	3)	TMR2 CH2		SSP1 CLK	
PD11	ADC11		PD11	TMR1 CH2N		TMR2 CH2N	SSP1 RXD		
PD12	ADC12		PD12	TMR1 CH3		TMR2 CH3	SSP1 TXD		
PD13	ADC13		PD13	TMR1 CH3N		TMR2 CH3N	CAN1 TX		
PD14	ADC14		PD14	TMR1 CH4		TMR2 CH4	CAN1 RX		
PD15	ADC15		PD15	CAN2 RX		4)	BUSY2	1)	EXT INT3

Рисунок 2.10 - Порты ввода-вывода – 4 часть

Порт E									
PE0	DAC2_OUT	6	PE0	ADDR16	1)	TMR2_CH1	15)	CAN1_RX	
PE1	DAC2_REF		PE1	ADDR17		TMR2_CH1N		CAN1_TX	
PE2	COMP_IN1	7	PE2	ADDR18		TMR2_CH3		TMR3_CH1	
PE3	COMP_IN2		PE3	ADDR19		TMR2_CH3N		TMR3_CH1N	
PE4	COMP_REF+		PE4	ADDR20		TMR2_CH4N		TMR3_CH2	
PE5	COMP_REF-		PE5	ADDR21		TMR2_BLK		TMR3_CH2N	
PE6	OSC_IN32	8	PE6	ADDR22		CAN2_RX	4)	TMR3_CH3	
PE7	OSC_OUT32		PE7	ADDR23		CAN2_TX		TMR3_CH3N	
PE8	COMP_IN3	7	PE8	ADDR24		TMR2_CH4	15)	TMR3_CH4	
PE9	DAC1_OUT		PE9	ADDR25		TMR2_CH2		TMR3_CH4N	
PE10	DAC1_REF	6	PE10	ADDR26		TMR2_CH2N		TMR3_ETR	
PE11	-		PE11	ADDR27		nSIRIN1		TMR3_BLK	
PE12	-		PE12	ADDR28		SSP1_RXD	16)	UART1_RXD	
PE13	-		PE13	ADDR29		SSP1_FSS		UART1_TXD	
PE14	-		PE14	ADDR30		TMR2_ETR	15)	SCL1	
PE15	-		PE15	ADDR31		EXT_INT3		SDA1	

Рисунок 2.11 - Порты ввода-вывода – 5 часть

Порт F									
PF0	-		PF0	ADDR0	1)	SSP1TXD	16)	UART2_RXD	14)
PF1	-		PF1	ADDR1		SSP1CLK		UART2_TXD	
PF2	-		PF2	ADDR2		SSP1FSS		CAN2_RX	
PF3	-		PF3	ADDR3		SSP1RXD		CAN2_TX	
PF4	-		PF4 MODE[0]	ADDR4		-		-	
PF5	-		PF5 MODE[1]	ADDR5		-		-	
PF6	-		PF6 MODE[2]	ADDR6		TMR1_CH1	3)	-	
PF7	-		PF7	ADDR7		TMR1_CH1N		TMR3_CH1	
PF8	-		PF8	ADDR8		TMR1_CH2		TMR3_CH1N	
PF9	-		PF9	ADDR9		TMR1_CH2N		TMR3_CH2	
PF10	-		PF10	ADDR10		TMR1_CH3		TMR3_CH2N	
PF11	-		PF11	ADDR11		TMR1_CH3N		TMR3_ETR	
PF12	-		PF12	ADDR12		TMR1_CH4		SSP2_FSS	
PF13	-		PF13	ADDR13		TMR1_CH4N		SSP2_CLK	
PF14	-		PF14	ADDR14		TMR1_ETR		SSP2_RXD	
PF15	-		PF15	ADDR15		TMR1_BLK		SSP2_TXD	

Рисунок 2.12 - Порты ввода-вывода – 6 часть

Выбираем порт ввода-вывода в таблице – Описание регистров портов ввода-вывода. (Рисунок 2.13)

Базовый Адрес	Название	Описание
0x400A 8000	MDR_PORTA	Порт A
0x400B 0000	MDR_PORTB	Порт B
0x400B 8000	MDR_PORTC	Порт C
0x400C 0000	MDR_PORTD	Порт D
0x400C 8000	MDR_PORTE	Порт E
0x400E 8000	MDR_PORTF	Порт F

Рисунок 2.13 - Описание регистров портов ввода-вывода

Находим адрес порта для нашего примера: Адрес физический - 0x400A_8000 - базовый адрес порта A – Функция для обращения к порту на языке Си в названии.

Находим название функции которую хотим осуществить на определенном порте в таблице – Описание регистров портов ввода-вывода. (рисунок 2.14)

Смещение			
0x00	RXTX[15:0]	MDR_PORTx->RXTX	Данные порта
0x04	OE[15:0]	MDR_PORTx->OE	Направление порта
0x08	FUNC[31:0]	MDR_PORTx->FUNC	Режим работы порта
0x0C	ANALOG[15:0]	MDR_PORTx->ANALOG	Аналоговый режим работы порта
0x10	PULL[31:0]	MDR_PORTx->PULL	Подтяжка порта
0x14	PD[31:0]	MDR_PORTx->PD	Режим работы выходного драйвера
0x18	PWR[31:0]	MDR_PORTx->PWR	Режим мощности передатчика
0x1C	GFEN[15:0]	MDR_PORTx->GFEN	Режим работы входного фильтра

Рисунок 2.14 - Описание регистров портов ввода-вывода

Для нашего примера нужно выбрать «Подтяжка порта»

PULL[31:0] – смещение 0x10

$0x400A_8000(\text{базовый адрес}) + 0x10(\text{смещение}) = 0x400A_8010$ - адрес PULL, он состоит из 32 бит.

0x400A_8010 - адрес PULL – Подтяжка порта

Название функции к которой нужно обратиться определена в названии – PULL, а пример обращения в колонке «описание» данной таблицы - MDR_PORTA->PULL

Для обращения к аналоговому порту с использованием определенного мода который мы определили что должны установить в начале, нужно обратиться к определенным функциям из таблицы. Обращаемся к аналоговому режиму порта по функции ANALOG по функции для Си MDR_PORTA->ANALOG

Адрес для ANALOG=0x400A_800C – Аналоговый режим порта

И должны выбрать мод «10» выбранный ранее. Для этого обращаемся к FUNC, функция для Си MDR_PORTA->FUNC

Адрес для FUNC=0x400A_8008 – Режим работы порта

Далее смотрим как выбранные нами функции определены (PULL, ANALOG, FUNC).

Описание функций (рисунок 2.15 – 2.19):

16.1.1 MDR_PORTx->RXTX

Таблица 124 – Регистр RXTX

Номер	31...16	15...0
Доступ	U	R/W
Сброс	0	0
	-	PORT RXTX[15:0]

Таблица 125 – Описание бит регистра RXTX

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31...16	-	Зарезервировано
15...0	PORT RXTX[15:0]	Режим работы контроллера. Данные для выдачи на выходы порта и для чтения

Рисунок 2.15 – Состав регистра RXTX

16.1.2 MDR_PORTx->OE

Таблица 126 – Регистр OE

Номер	31...16	15...0
Доступ	U	R/W
Сброс	0	0
	-	PORT OE[15:0]

Таблица 127 – Описание бит регистра OE

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31...16	-	Зарезервировано
15...0	PORT OE[15:0]	Режим работы контроллера. Направление передачи данных на выводах порта: 1 – выход; 0 – вход

Рисунок 2.6 - Состав регистра OE

16.1.3 MDR_PORTx->FUNC

Таблица 128 – Регистр FUNC

Номер	31	30	...	3	2	1	0
Доступ	R/W	R/W	...	R/W	R/W	R/W	R/W
Сброс	0	0	...	0	0	0	0
	MODE15[1:0]		...	MODE1[1:0]		MODE0[1:0]	

Таблица 129 – Описание бит регистра FUNC

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31...2	MODEx	Аналогично MODE0 для остальных бит порта
1...0	MODE0[1:0]	Режим работы вывода порта: 00 – порт; 01 – основная функция; 10 – альтернативная функция 11 – переопределенная функция

Рисунок 2.17 - Состав регистра FUNC

Из нашего задания FUNC отвечает за MODE который нам нужно установить в «10»-альтернативные функции. Нам нужно выбрать один из 32 MODEx к которому будем обращаться, это MODE1. Далее устанавливаем MODE1 в 10.

Адрес для FUNC=0x400A_8008 – Режим работы порта – начало адресации FUNC

Для MODE0 первые 2 бита отвечают за режим, и таким образом по два бита из 32 отведенных для MODEx отводится на каждый отдельный MODE1, MODE2 и тд.

Нам нужен MODE1 – значит 00[11]000000000000000000000000000000 – 2 и 3 биты нужно установить в единицу.

16.1.4 MDR_PORTx->ANALOG

Таблица 130 – Регистр ANALOG

Номер	31...16	15...0
Доступ	U	R/W
Сброс	0	0
	-	ANALOG EN[15:0]

Таблица 131 – Описание бит регистра ANALOG

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31...16	-	Зарезервировано
15...0	ANALOG EN[15:0]	Режим работы контроллера: 0 – аналоговый; 1 – цифровой

Рисунок 2.18 - Состав регистра ANALOG

Из нашего задания ANALOG нужно установить в 1, цифровой.

ANALOG=0x400A_800C – Аналоговый режим порта – Начало адресации ANALOG

Всего ANALOG состоит из 32 бит и 16-31 – отведены под резерв, а 0-15 отвечают за порт с которым мы собираемся работать. Нам нужен PA1 первый порт из первой таблицы, значит во второй по порядку бит устанавливаем в единицу.

16.1.5 MDR_PORTx->PULL

Таблица 132 – Регистр PULL

Номер	31...16	15...0
Доступ	R/W	R/W
Сброс	0	0
	PULL UP[15:0]	PULL DOWN[15:0]

Таблица 133 – Описание бит регистра PULL

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31...16	PULL UP[15:0]	Режим работы контроллера. Разрешение подтяжки вверх: 0 – подтяжка в питание выключена; 1 – подтяжка в питание включена (есть подтяжка ~50 кОм)
15...0	PULL DOWN[15:0]	Режим работы контроллера. Разрешение подтяжки вниз: 0 – подтяжка в ноль выключена; 1 – подтяжка в ноль включена (есть подтяжка ~ 50 кОм)

Рисунок 2.19 - Состав регистра PULL

0x400A_8010 - адрес PULL – Подтяжка порта

Здесь на PULL отводится 32 бита – первые 16 бит отвечают за PULL_DOWN (подтяжка на землю), а остальные 16 за PULL_UP (подтяжка по питанию). Каждый от 0-15 бит отвечает за порт к которому мы будем обращаться.

Нам нужен первый порт и сделать подтяжку по питанию, значит – PULL_UP и нужен порт PA1 значит второй по счету порт и это 17 бит (т.к. в рамках 16-31 и второй по счету PA0, PA1, ..). (Рисунок 2.20 – 2.22)

16.1.6 MDR_PORTx->PD

Таблица 134 – Регистр PD

Номер	31...16	15...0
Доступ	R/W	R/W
Сброс	0	0
	PORT SHM[15:0]	PORT PD[15:0]

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31...16	PORT SHM[15:0]	Режим работы контроллера. Режим работы входа: 0 – триггер Шмитта выключен гистерезис 200 мВ; 1 – триггер Шмитта включен гистерезис 400 мВ
15...0	PORT PD[15:0]	Режим работы контроллера. Режим работы выхода: 0 – управляемый драйвер; 1 – открытый сток

Рисунок 2.20 - Состав регистра PD

16.1.7 MDR_PORTx->PWR

Таблица 136 – Регистр PWR

Номер	31	30	...	3	2	1	0
Доступ	R/W	R/W	...	R/W	R/W	R/W	R/W
Сброс	0	0	...	0	0	0	0
	PWR15[1:0]		...	PWR1[1:0]		PWR0[1:0]	

Таблица 137 – Описание бит регистра PORTx_PWR

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31...2	PWRx	Аналогично PWR0 для остальных бит порта
1...0	PWR0[1:0]	Режим работы вывода порта: 00 – зарезервировано (передатчик отключен) 01 – медленный фронт (порядка 100 нс) 10 – быстрый фронт (порядка 20 нс) 11 – максимально быстрый фронт (порядка 10 нс)

Рисунок 2.21 - Состав регистра PWR

16.1.8 MDR_PORTx->GFEN

Таблица 138 – Регистр GFEN

Номер	31...16	15...0
Доступ	R/W	R/W
Сброс	0	0
	-	GFEN[15:0]

Таблица 139 – Описание бит регистра GFEN

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31...16	-	Зарезервировано
15...0	GFEN[15:0]	Режим работы входного фильтра: 0 – фильтр выключен; 1 – фильтр включен (фильтрация импульсов до 10 нс)

Рисунок 2.22 - Состав регистра GFEN

Для инициализации используется структура типа `PORT_InitTypeDef`, поэтому необходимо объявить переменную данного типа. (Рисунок 3.2 – 3.4)

```
//Функция инициализации кнопки SA4
void button_Init(void){
RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD, ENABLE); //включить тактирование порта D
//Создание структуры для инициализации порта
PORT_InitTypeDef PORT_InitStructure;
//Настройки порта: ввод, функция ввода/вывода, цифровой режим, минимальная скорость,
Pin5
PORT_InitStructure.PORT_OE = PORT_OE_IN; //Конфигурирование порта - ввод
PORT_InitStructure.PORT_FUNC = PORT_FUNC_PORT; //Режим работы порта - порт
PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL; //Режим работы контроллера -
цифровой
PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW; //Скорость работы - медленно
PORT_InitStructure.PORT_Pin = (PORT_Pin_5); //Выбор выводов для инициализации - номер
вывода порта - Pin_5

PORT_Init(MDR_PORTD, &PORT_InitStructure); //Передача структуры порту D
}
```

Рисунок 3.2 – Код конфигурирования кнопки – конфигурирование порта на ввод

```
//Функция инициализации светодиода VD7
void led_Init(void){
RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE); //включить тактирование порта C

//Создание структуры для инициализации порта
PORT_InitTypeDef PORT_InitStructure; //Объявляем структуру для конфигурации порта

//Настройки порта: вывод, функция ввода/вывода, цифровой режим, максимальная скорость,
Pin2
PORT_InitStructure.PORT_OE = PORT_OE_OUT; //Направление порта - вывод
PORT_InitStructure.PORT_FUNC = PORT_FUNC_PORT; //Режим работы порта - порт
PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL; //Режим работы контроллера -
цифровой
PORT_InitStructure.PORT_SPEED = PORT_SPEED_MAXFAST; //Скорость работы -
максимально быстро
PORT_InitStructure.PORT_Pin = (PORT_Pin_2); //Выбор выводов для инициализации - номер
вывода порта - Pin_2

PORT_Init(MDR_PORTC, &PORT_InitStructure); //Передача структуры порту C
}
```

Рисунок 3.3 – Код конфигурирования светодиода – конфигурирование порта на вывод

```
//функция считывания текущего состояния кнопки SA4
uint8_t button_State(void){
    return PORT_ReadInputDataBit(MDR_PORTD, PORT_Pin_5);
}

//функция записи состояния (1:0) светодиода VD7
void led_Write(bool on_off){
    PORT_WriteBit(MDR_PORTC, PORT_Pin_2, on_off ? Bit_SET : Bit_RESET);
}
```

Рисунок 3.4 – Функции работы с кнопкой и светодиодом

Подключить кнопку и светодиод к соответствующим портам. Нажатие кнопки должно зажигать (или гасить) светодиод.

Чтобы светодиод загорался при нажатии на кнопку необходимо прописать на базе имеющихся функций следующую конструкцию. (Рисунок 3.5)

```
int main(){
    led_Init();
    button_Init();
    uint8_t state = 0;
    printf("Start cycle\n");
    while (1){
        //Считывание текущего состояния кнопки SA4
        state = button_State(); //Считать состояние SA4
        led_Write(!state); //Включить светодиод при нажатии SA4
    }
}
```

Рисунок 3.5 – Код для мигания светодиодом по нажатию на кнопку

Далее, был написан код для общения двух мекроконтроллеров посредством коротких импульсов и кодами азбуки морзе.

На рисунке 3.6 отображена схема объединения двух плат посредством интерфейса GPIO. Соединение, отмеченное черным цветом, образует общую землю узлов «сети», по остальным 4 проводам происходит обмен информацией между узлами. Из них два провода предназначены для передачи стробирующих импульсов, другие два – для передачи и приёма

закодированной

информации.

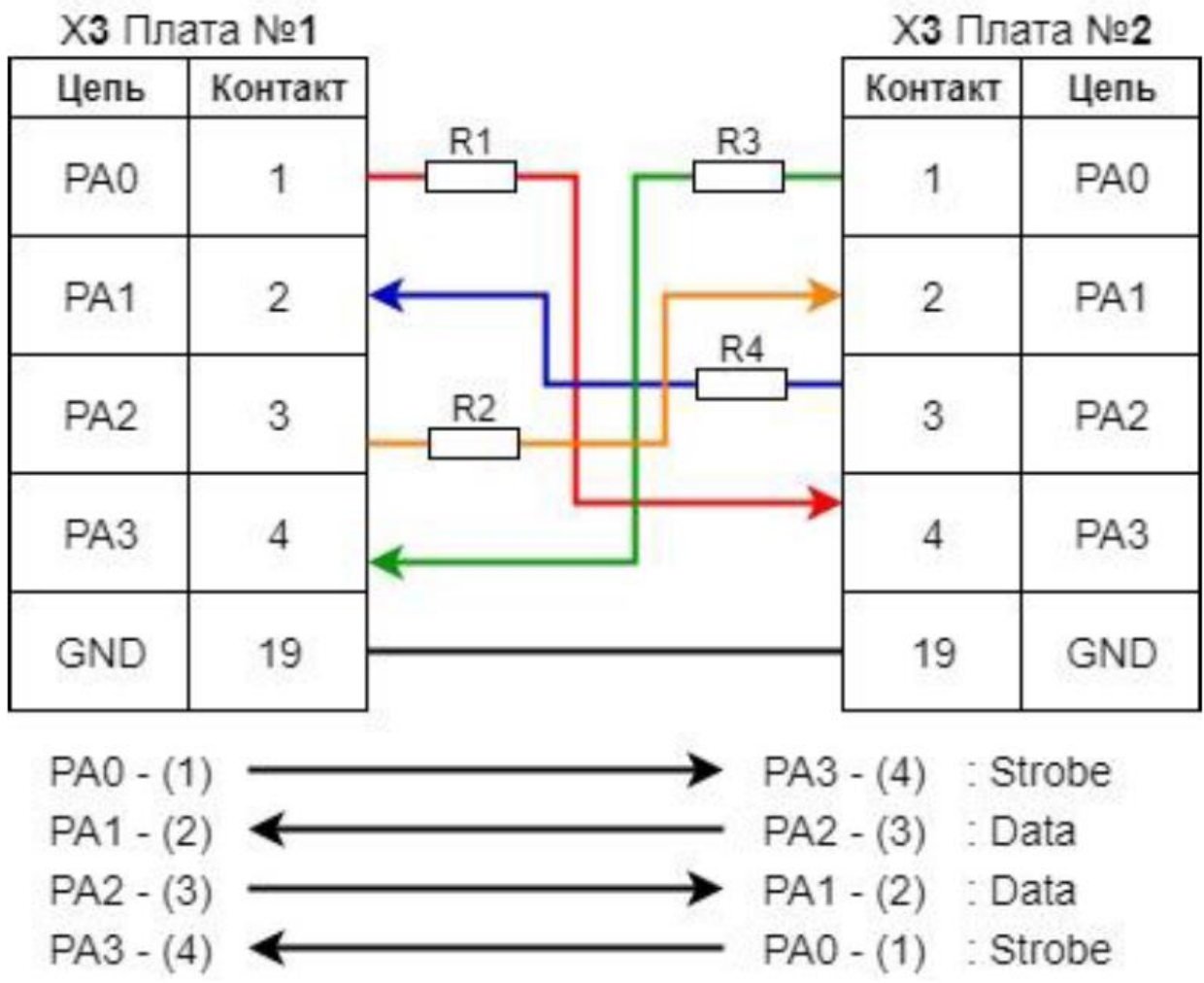


Рисунок 3.6 – Реализованная схема соединения плат

Резисторы R1-R4 необходимо использовать для защиты от короткого замыкания.

Если соединить два порта, настроенных на вывод, разных плат, то упрощенно получается схема Н-моста. На рисунке 3.7 представлен режим работы, при котором ток идёт через нагрузку. В этом случае короткого замыкания возникнуть не может.

В случае соединения микроконтроллеров в качестве «Load» необходимо использовать резистор.

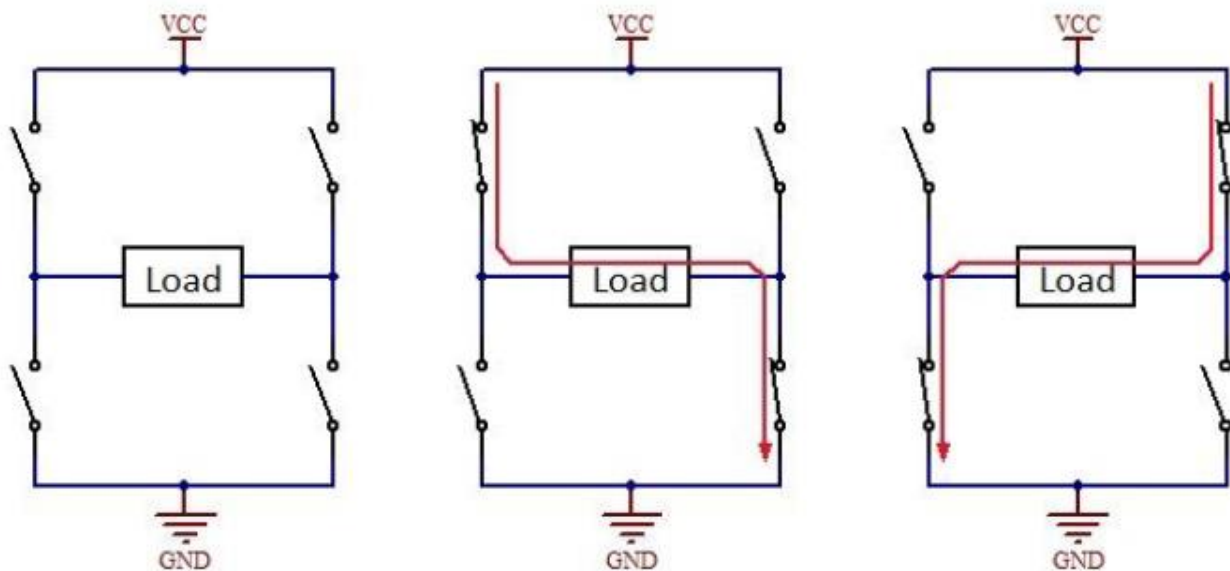


Рисунок 3.7 – Название

После запуска программ, микроконтроллеры переходят в режим ожидания приёма сигнала. В режиме ожидания пользователю доступна возможность набора сообщений на азбуке Морзе с помощью кнопки на плате. Единичное нажатие кнопки формирует точку, нажатие с удержанием в несколько секунд формирует тире. В одном сообщении можно отправить не более восьми знаков, в любом порядке. Таким образом сообщение может либо соответствовать латинской букве или цифре (согласно азбуке), либо не соответствовать. После написания, сообщение можно отправить, удерживая кнопку до появления сообщения в окне программы отправителя: «The END».

После отпускания кнопки начинается процедура знакового кодирования и отправки. На МК получателя, находящегося в режиме ожидания, с МК отправителя по каналу PA0→PA3 отправляется строб, сигнализирующий МК получателя, о начале процедуры приёма сигнала с вывода PA1. После установки строба каждый знак введённого сообщения с задержкой в 200 мс устанавливается на выводе PA2 МК отправителя в виде высокого или низкого уровня напряжения (высокий - точка, низкий - тире) и передаётся с него на вывод PA1 МК получателя. МК получателя с такой же задержкой распознаёт каждый пришедший знак и записывает его в буферный массив.

Когда МК получателя перестаёт получать строб, заканчивается процедура приёма и начинается процедура распознавания символа. Содержимое буферного массива сравнивается с последовательностями знаков, соответствующих буквам латинского алфавита и цифрам. Если в «словаре» есть последовательность знаков, идентичная принятой, то в консоли программы отображается распознанный символ, иначе программа сообщает получателю, что сообщение не распознано. Принятая последовательность знаков так же отображается посредством светодиода на плате. Светодиод отображает каждый знак принятого сообщения

– он загорается на 0,5 секунды, если получена точка, и на 1,5 секунды, если получено тире, между знаками сообщения светодиод гаснет на 0,5 секунды.

На рисунках 3.8-3.10 изображены текстовые сообщения, отображаемые в консоли программы, при передаче сообщений между МК, на стороне отправителя и получателя.

Незаполненные знаки в сообщениях помечаются символом «X», который так же МК определить конец сообщения, если оно состоит меньше чем из 8 знаков.

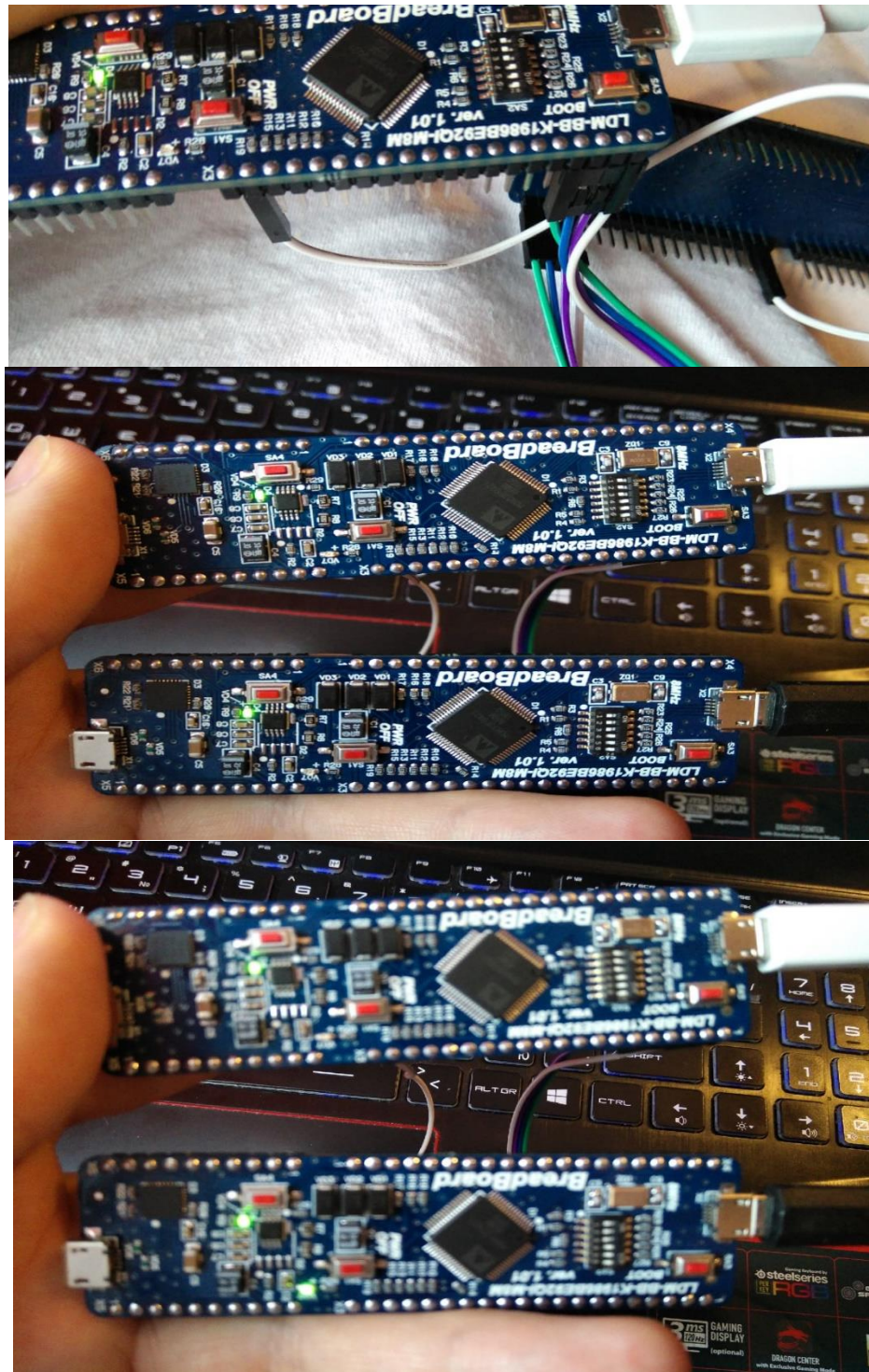


Рисунок 3.8 – Подключение устройств

```

[11:55:19] DEBUG INFO: --.....
[11:55:19] Start transmit symbol...
[11:55:20] End transmit symbol!
[11:55:20] Char: M
[11:55:25] DEBUG INFO: -ZZZZZZ
[11:55:26] DEBUG INFO: --ZZZZZ
[11:55:27] DEBUG INFO: ---ZZZZ
[11:55:29] DEBUG INFO: ---.ZZZ
[11:55:29] DEBUG INFO: ---..ZZ
[11:55:30] DEBUG INFO: ---...Z
[11:55:31] DEBUG INFO: ---....
[11:55:33] DEBUG INFO: ---....
[11:55:33] Start transmit symbol...|
[11:55:34] End transmit symbol!
[11:55:34] Char: O
[11:56:21] DEBUG INFO: .-.....-
[11:56:21] Start transmit symbol...
[11:56:23] End transmit symbol!
[11:56:23] Char: R
[11:56:53] DEBUG INFO: --.-....
[11:56:53] Start transmit symbol...
[11:56:55] End transmit symbol!
[11:56:55] Char: Z
[11:57:04] DEBUG INFO: .....
[11:57:04] Start transmit symbol...
[11:57:06] End transmit symbol!
[11:57:06] Char: E

```

Рисунок 3.9 – Передача сообщений

```

[11:58:02] Start receive symbol!
[11:58:04] End receive symbol!
[11:58:04] Sign: --.....
[11:58:20] Char: M
[11:58:20] Start receive symbol!
[11:58:21] End receive symbol!
[11:58:21] Sign: ---.....
[11:58:39] Char: O
[11:58:54] Start receive symbol!
[11:58:56] End receive symbol!
[11:58:56] Sign: .-.....-.
[11:59:12] Char: R
[11:59:14] Start receive symbol!
[11:59:16] End receive symbol!
[11:59:16] Sign: --.-.....
[11:59:34] Char: Z
[11:59:48] Start receive symbol!
[11:59:50] End receive symbol!
[11:59:50] Sign: .....
[12:00:02] Char: E

```

Рисунок 3.10 – Прием сообщений

По причине выполнения лабораторной работы в домашних условиях, без использования специального оборудования, привести изображение с экрана осциллографа, при передаче сообщений между МК, не представляется возможным.

3 Заключение

В ходе выполнения лабораторной работы была составлена программа, позволяющая пользователю с помощью азбуки Морзе сформировать сообщение и реализующая процедуры передачи и приёма этих сообщений посредством интерфейса GPIO.

Для проверки работы программы была организована сеть из 2 микроконтроллеров K1986BE92QI (на платах от LDM-Systems), обменивающихся информацией через GPIO. Для создания программы использовалась IDE NetBeans.

Был написан отчёт согласно требованиям ОС ТУСУР 01-2013.

Приложение А

(обязательное)

Код – main

```

/*+---<INCLUDES> -----*/

#include "main.h"
#include <stdlib.h>
#include <stdio.h>
//#include "stdint.h"
#include <inttypes.h>


#include "MDR32F9Qx_board.h"
#include "MDR32F9Qx_config.h"
#include "MDR32Fx.h"
#include "MDR32F9Qx_port.h"
#include "MDR32F9Qx_rst_clk.h"
#include "mstn_led.h"
#include "mstn_clk.h"
#include "mstn_usb.h"


/*+======*/
/*+---<DEFINES> -----*/
#define LED1 PORT_Pin_0
/*+======*/
/*+---<FUNCTIONS DECLARATION> -----*/


void clk_CoreConfig(void);


//Функция инициализации кнопки SA4
void button_Init(void);

//Функция считывания текущего состояния кнопки SA4
uint8_t button_State(void);


//Функция инициализации светодиода VD7

```

```

void led_Init(void);
//Функция записи состояния (1:0) светодиода VD7
void led_Write(bool on_off);
void BlinkLED(uint32_t num, uint32_t del);
void IndicateError(void);

//Morze
uint8_t MorzeToChar (uint8_t *symb);
void ClearSymbol(uint8_t *symb);
void LedMorze(uint8_t *symbols);
void PinInitA(PORT_OE_TypeDef mode, uint16_t pin);
//Rx Tx
void TransmitSymbol (uint8_t *symb);
void ReceiveSymbol(uint8_t *symb);

//int main(int argc, char *argv[])
int main()
{
    /*
    printf("Usb cycle\n");
    while(USB_GetStatus() != PERMITTED);
    printf("USB ready\n");

    printf("[");
    for(int i=0;i<100;i++){
        printf("%d",i);
        printf(":");
    }
    printf("]\n");
    */
    //clk_CoreConfig();
    led_Init();
    button_Init();
    PinInitA(PORT_OE_OUT, PORT_Pin_2);
    PinInitA(PORT_OE_IN, PORT_Pin_1);

```

```

PinInitA(PORT_OE_OUT, PORT_Pin_0);
PinInitA(PORT_OE_IN, PORT_Pin_3);

//printf("Buffer save\n");
BlinkLED(5,300);
//printf("end\n");

//uint8_t state = 0;
//uint32_t waitTime = 5;
uint8_t counter = 0;
uint8_t symbol_tx[8];
uint8_t symbol_rx[8];
uint8_t numbBit = 0;
uint8_t tempValue = 255;

while (1){

    /*
    //printf("Input waitTime (ms):\n");
    //scanf("%lu", &waitTime);

    printf("Buffer save\n");
    BlinkLED(4,10000);
    printf("end\n");
    */

    if (PORT_ReadInputDataBit(MDR_PORTA, PORT_Pin_3)) {
        ClearSymbol(symbol_rx);
        ReceiveSymbol(symbol_rx);
        LedMorze(symbol_rx);
        tempValue = MorzeToChar(symbol_rx);
    }
}

```

```

tempValue == 255 ? printf("Symbol is undefined! \n") : printf("Char: %c \n",
tempValue);
}
// Read button
while (!button_State()) {
    counter++;
    Delay(500);
    if (counter == 4) printf("DEBUG INFO: Completed char! \n");
}
// Define sign
if (counter > 0) {
    if (counter == 1) {
        symbol_tx[numbBit] = '.';
        numbBit++;
        printf("DEBUG INFO: %s \n", symbol_tx);
    }
    else if (counter < 4) {
        symbol_tx[numbBit] = '-';
        numbBit++;
        printf("DEBUG INFO: %s \n", symbol_tx);
    }
    else{
        for(numbBit;numbBit>8;--numbBit){
            symbol_tx[numbBit] = 'x';
        }
    }
// If end symbol
if (numbBit == 8) {
    numbBit = 0;
    TransmitSymbol(symbol_tx);
    tempValue = MorzeToChar(symbol_tx);
    tempValue == 255 ? printf("Symbol is undefined! \n") : printf("Char: %c \n",
tempValue);
}
}
}

```

```

    if (counter != 0) {
        //printf("numbSymb: %d \n", numbBit);
    }
    counter = 0;

}
/*
printf("Start cycle\n");
while (1){
    //Считывание текущего состояния кнопки SA4
    state = button_State(); //Считать состояние SA4
    led_Write(!state); //Включить светодиод при нажатии SA4
    printf("%d",state);
}
*/
return EXIT_SUCCESS;
}

/*+---<FUNCTIONS DESCRIPTION> -----*/
//
void BlinkLED(uint32_t num, uint32_t del){
    uint32_t cnt;
    for ( cnt = 0; cnt < num; cnt++)
    {
        Delay(del);
        led_Write(1);
        Delay(del);
        led_Write(0);
    }
}
//
void IndicateError(void){
    /*<<<>>> Switch LED3 on and off in case of error */
    BlinkLED(3,5000);
}

```

```
}
```

```
//Функция инициализации кнопки SA4
```

```
void button_Init(void){
```

```
//Создание структуры для инициализации порта
```

```
PORT_InitTypeDef PORT_InitStructure;
```

```
//Настройки порта: ввод, функция ввода/вывода, цифровой режим,
```

```
//минимальная скорость, Pin5
```

```
PORT_InitStructure.PORT_OE = PORT_OE_IN;
```

```
PORT_InitStructure.PORT_FUNC = PORT_FUNC_PORT;
```

```
PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;
```

```
PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
```

```
PORT_InitStructure.PORT_Pin = (PORT_Pin_5);
```

```
PORT_Init(MDR_PORTD, &PORT_InitStructure);
```

```
}
```

```
//Функция считывания текущего состояния кнопки SA4
```

```
uint8_t button_State(void){
```

```
return PORT_ReadInputDataBit(MDR_PORTD, PORT_Pin_5);
```

```
}
```

```
//Функция инициализации светодиода VD7
```

```
void led_Init(void){
```

```
//Создание структуры для инициализации порта
```

```
PORT_InitTypeDef PORT_InitStructure;
```

```
//Настройки порта: вывод, функция ввода/вывода, цифровой режим,
```

```
//максимальная скорость, Pin2
```

```
PORT_InitStructure.PORT_OE = PORT_OE_OUT;
```

```
PORT_InitStructure.PORT_FUNC = PORT_FUNC_PORT;
```

```
PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;
```

```
PORT_InitStructure.PORT_SPEED = PORT_SPEED_MAXFAST;
```

```
PORT_InitStructure.PORT_Pin = (PORT_Pin_2);
```

```
PORT_Init(MDR_PORTC, &PORT_InitStructure);
```

```
}
```

```
//Функция записи состояния (1:0) светодиода VD7
```

```
void led_Write(bool on_off){
```

```

PORT_WriteBit(MDR_PORTC, PORT_Pin_2, on_off ? Bit_SET : Bit_RESET);
}

//Функция настройки тактовой частоты МК
void clk_CoreConfig(void) {
    //Реинициализация настроек тактирования
    // Включить тактирование батарейного блока
    //и внутренние генераторы, все остальное сбросить
    RST_CLK_DeInit();
    //Включение тактирования от внешнего источника HSE (High Speed External)
    RST_CLK_HSEconfig(RST_CLK_HSE_ON);
    //Проверка статуса HSE
    //if (RST_CLK_HSEstatus() == SUCCESS) /* Если HSE осциллятор включился
    //if (RST_CLK_HSEstatus() == ERROR) while (1);
    while (RST_CLK_HSEstatus () != SUCCESS);
    //Настройка делителя/умножителя частоты CPU_PLL(фазовая подстройка
частоты)
    /* Указываем PLL от куда брать частоту (RCC_PLLSource_HSE_Div1) и на
сколько ее умножать (RCC_PLLMul_9) */
    /* PLL может брать частоту с кварца как есть (RCC_PLLSource_HSE_Div1) или
поделенную на 2 (RCC_PLLSource_HSE_Div2). Смотри схему */
    RST_CLK_CPU_PLLconfig(RST_CLK_CPU_PLLsrcHSEdiv1,
RST_CLK_CPU_PLLmul10);
    // RST_CLK_CPU_PLLconfig(div, mul);
    //Включение CPU_PLL
    //, но еще не подключать к кристаллу
    RST_CLK_CPU_PLLcmd(ENABLE);
    //Проверка статуса CPU_PLL
    //if (RST_CLK_CPU_PLLstatus() == SUCCESS) //Если включение CPU_PLL
прошло успешно
    //if (RST_CLK_CPU_PLLstatus() == ERROR) while (1);
    while (RST_CLK_CPU_PLLstatus() != SUCCESS);

    /* Установка CPU_C3_prescaler = 2 */
    // Делитель CPU_C3_SEL ( CPU_C3_SEL = CPU_C2_SEL/2 )

```



```

RST_CLK_CPUclkPrescaler(RST_CLK_CPUclkDIV1);

//Коммутация выхода CPU_PLL на вход CPU_C3
//На C2 идет с PLL, а не напрямую с C1 (CPU_C2_SEL = PLL)
RST_CLK_CPU_PLLuse(ENABLE);
//Выбор источника тактирования ядра процессора
//CPU берет с выхода C3 (а может с выхода HSI,LSI,LSE) (HCLK_SEL =
CPU_C3_SEL )
RST_CLK_CPUclkSelection(RST_CLK_CPUclkCPU_C3);
//Тактирование периферии
//Подача тактовой частоты на PORTC, PORTD
RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTD, ENABLE);
    SystemCoreClockUpdate();
}

void PinInitA(PORT_OE_TypeDef mode, uint16_t pin){
    PORT_InitTypeDef PORT_InitStructure;
    PORT_InitStructure.PORT_OE = mode;
    PORT_InitStructure.PORT_FUNC = PORT_FUNC_PORT;
    PORT_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;
    PORT_InitStructure.PORT_SPEED = PORT_SPEED_SLOW;
    PORT_InitStructure.PORT_Pin = (pin);
    PORT_Init(MDR_PORTA, &PORT_InitStructure);
}

void LedMorze (uint8_t *symbols) {
    for (uint8_t i = 0; i < 8; ++i) {
        if (symbols[i] == '.') {
            BlinkLED(1,500);
            //Delay(500);
        }
        else if (symbols[i] == '-') {

```

```

        BlinkLED(1,1500);
    }
    else {
        led_Write(0);
        break;
    }
    led_Write(0);
    Delay(500);
}
}

```

Код – RxDx

```

#include "MDR32F9Qx_config.h"
#include "MDR32F9Qx_board.h"
#include "MDR32F9Qx_rst_clk.h"
#include "MDR32F9Qx_port.h"
#include <stdint.h>
#include <stdio.h>
#include "mstn_clk.h"

//Rx Tx
void TransmitSymbol (uint8_t *symb) {

    printf("Start transmit symbol...\n");

    PORT_SetBits(MDR_PORTA, PORT_Pin_0);

    for(uint8_t i = 0; i < 8; i++) {
        if(symb[i] == '.')
            PORT_SetBits(MDR_PORTA, PORT_Pin_2);
        else if(symb[i] == '-')
            PORT_ResetBits(MDR_PORTA, PORT_Pin_2);
        else

```

```

        break;
    Delay(200);
}
PORT_ResetBits(MDR_PORTA, PORT_Pin_0);
printf("End transmit symbol!\n");
PORT_ResetBits(MDR_PORTA, PORT_Pin_2);
}

void ReceiveSymbol(uint8_t *symb) {
    printf("Start receive symbol!\n");

    uint8_t i = 0;

    while(PORT_ReadInputDataBit(MDR_PORTA, PORT_Pin_3))
    {
        if(PORT_ReadInputDataBit(MDR_PORTA, PORT_Pin_1) != 0)
            symb[i] = '.';
        else
            symb[i] = '-';
        i++;
        Delay(200);
    }
    printf("End receive symbol!\n");
    printf("Sign: %s \n", symb);
}

```

Код – Morze

```

/*
#include "MDR32F9Q._config.h"
#include "MDR32F9Q._board.h"
#include "MDR32F9Q._rst_clk.h"
#include "MDR32F9Q._port.h"
*/#include <stdint.h>
#include <stdio.h>
#include "mstn_clk.h"

```

```
uint8_t MorzeToChar (uint8_t *symb);
```

```
void ClearSymbol (uint8_t *symb);
```

```
uint8_t AlphabetMorze [] = {'.', '.', '.', '.', '.', '.', '.', '.', // E 0
```

```
    '-', '.', '.', '.', '.', '.', '.', '.', // T 1
```

```
    '.', '.', '.', '.', '.', '.', '.', '.', // I 2
```

```
    '-', '-', '.', '.', '.', '.', '.', '.', // M 3
```

```
    '.', '.', '.', '.', '.', '.', '.', '.', // S 4
```

```
    '-', '-', '-', '.', '.', '.', '.', '.', // O 5
```

```
    '.', '.', '.', '.', '.', '.', '.', '.', // H 6
```

```
    '-', '.', '.', '.', '.', '.', '.', '.', // N 7
```

```
    '.', '-', '.', '.', '.', '.', '.', '.', // A 8
```

```
    '-', '-', '.', '.', '.', '.', '.', '.', // G 9
```

```
    '.', '.', '-', '.', '.', '.', '.', '.', // U 10
```

```
    '-', '-', '.', '-', '.', '.', '.', '.', // Z 11
```

```
    '.', '.', '.', '-', '.', '.', '.', '.', // V 12
```

```
    '-', '-', '.', '-', '.', '.', '.', '.', // Q 13
```

```
    '.', '-', '-', '.', '.', '.', '.', '.', // W 14
```

```
    '-', '.', '.', '.', '.', '.', '.', '.', // D 15
```

```
    '.', '-', '-', '-', '.', '.', '.', '.', // J 16
```

```
    '-', '.', '.', '.', '.', '.', '.', '.', // B 17
```

```
    '.', '-', '.', '.', '.', '.', '-', '.', // R 18
```

```
    '-', '.', '-', '.', '.', '.', '.', '.', // K 19
```

```
    '.', '-', '.', '.', '.', '.', '.', '.', // L 20
```

```
    '-', '.', '-', '.', '.', '.', '.', '.', // C 21
```

```
    '.', '.', '-', '.', '.', '.', '.', '.', // F 22
```

```
    '-', '.', '-', '-', '.', '.', '.', '.', // Y 23
```

```
    '.', '-', '-', '.', '.', '.', '.', '.', // P 24
```

```
    '-', '.', '.', '-', '.', '.', '.', '.', // X 25
```

```
    '.', '-', '-', '-', '-', '.', '.', '.', // 1 26
```

```
    '.', '.', '-', '-', '-', '.', '.', '.', // 2 27
```

```
    '.', '.', '.', '-', '-', '.', '.', '.', // 3 28
```

```
    '.', '.', '.', '.', '-', '.', '.', '.', // 4 29
```

```
    '.', '.', '.', '.', '.', '.', '-', // 5 30
```

```
    '-', '.', '.', '.', '.', '.', '-', // 6 31
```

```

        '-', '-', '!', '!', '!', '!', '!', '!', // 7 32
        '-', '-', '-', '!', '!', '!', '!', '!', // 8 33
        '-', '-', '-', '-', '!', '!', '!', '!', // 9 34
        '-', '-', '-', '-', '-', '!', '!', '!', // 0 35
        '!', '!', '!', '!', '!', '!', '!', '!', // SPACE 36
        '-', '!', '!', '!', '!', '!', '!', '!' // ! 37
};

uint8_t AlphabetEN[] = {'E', 'T', 'I', 'M', 'S', 'O', 'H', 'N',
                        'A', 'G', 'U', 'Z', 'V', 'Q', 'W', 'D',
                        'J', 'B', 'R', 'K', 'L', 'C', 'F', 'Y',
                        'P', 'X', '1', '2', '3', '4', '5', '6',
                        '7', '8', '9', '0', ' ', '!'};

};

//Morze
uint8_t MorzeToChar (uint8_t *symb) {
    bool flagCheckSymbol = false;
    uint8_t numbSymb = 255;
    uint8_t sizeArray = sizeof(AlphabetEN)/sizeof(uint8_t);
    for (uint8_t i = 0; i < sizeArray; ++i) {
        flagCheckSymbol = false;
        for (uint8_t j = 0; j < 8; ++j) {
            if (symb[j] == AlphabetMorze[j+(i*8)])
                flagCheckSymbol = true;
            else {
                flagCheckSymbol = false;
                break;
            }
        }
    }
    if (flagCheckSymbol == true) {

        numbSymb = i;
        break;
    }
}

```

```
}
```

```
ClearSymbol(symb);
```

```
if (numbSymb != 255) {
    for (uint8_t i = 0; i < sizeArray; ++i)
    {
        if (i == numbSymb) return AlphabetEN[i];
    }
}
else return 255;
//return 0;
}
```

```
void ClearSymbol(uint8_t *symb) {
    for (uint8_t i = 0; i < 8; ++i)
        symb[i] = '0';
}
```