

Serial Communication

1 Objectives

The objective of this laboratory exercise is to provide an overview of serial communication and demonstrate how it applies to the MRK board. The concept of an interrupt is introduced.

2 Theoretical Background

Parallel and serial communication are two different ways of transferring data. In parallel communication, all 8 bits are sent at the same time if one byte is being sent. Each bit has its own wire, and additional wires are required to coordinate the communication. Usually, at least 11 wires are required for 8-bit transfers, and 19 wires are required for 16-bit transfer. Parallel communication is very fast, but providing all of the wires and I/O pins can be expensive. An alternative is serial communication, which sends the bits sequentially over a single wire. Serial communication requires at least 2 wires: one for the signal and one for a common ground.

There are two methods of serial communication: synchronous and asynchronous. The MRK board can do both, but we will mostly be using asynchronous serial communication. Synchronous communication uses a clock signal to synchronize data transfer. One device generates the clock signal and sends it over a wire to the other device.

In asynchronous communication, each device generates its own clock (they must agree on the transmission speed) and additional bits are used to synchronize the transfer. The MRK board is configured to use one start bit and one stop bit, so each 8-bit data transfer requires 10 bits. The data transmission format is given in Figure 1. The start bit is a logical 0 and the stop bit is a logical 1. Note that the least significant bit D0 is sent first. The transmission speed in bits per second is called the baud rate. We will be using the default baud rate of 9600. Both the MRK board and the MRK terminal must be set to use the same baud rate. At 9600 baud, transmission of a single byte (8 bits) takes

$$(1 \text{ byte}) \times (10 \text{ bits / byte}) \times \left(\frac{1 \text{ second}}{9600 \text{ bits}} \right) = 0.001 \text{ s} = 1 \text{ ms}$$

One millisecond might sound small, but it is a very long time compared to the speed at which the MRK board can operate.

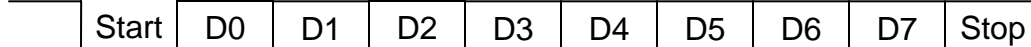
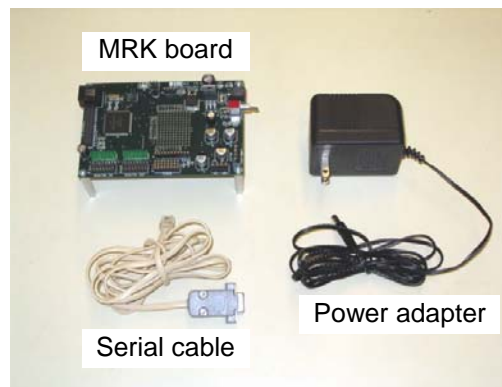


Figure 1: Data transmission format

When one interacts with a computer, he/she needs a way of storing the characters pictured on the keyboard as digital data. The ASCII character code defines the bit patterns which represent each character. An ASCII character code table is provided for reference at the end of the lab handout. In practice, the compiler will take care of the conversion for you. Enclose a character in single quotes (' z ') and the compiler will replace it with the appropriate digital representation.

An interrupt is a signal to the CPU that something has happened. Interrupts are useful because it is inefficient for the CPU to continually check whether an event has occurred. Imagine that you are trying to do your homework, but you are also expecting an important phone call. The ringer on the phone is broken, so every 5 seconds you pick up the phone to check if someone is on the line. The ringer on the phone is like an interrupt – it lets you know that someone is calling. You have different interrupts for different things: doorbell, oven timer, etc. We will look at the SCI interrupt in this lab and some of the other interrupts in later labs.

3 Equipment



- MRK board
- Serial communication cable
- Power adapter
- Oscilloscope

4 Procedures

4.1 Transmitting data

The header file “MRK.h” defines various functions that make serial communication rather simple. The function `fputchar(interface, character)` sends a single character (1 byte). The first argument chooses the interface. We will be using SCI interface 0 (SCI0). The second

argument is the character to be sent. Open Eclipse and create a new project. Add the following line to the main block:

```
fputcchar(SCI0, 'z');
```

Build, download, and run the program. To print an entire message, you may use a series of `fputcchar()` commands:

```
fputcchar(SCI0, 'h');
fputcchar(SCI0, 'e');
fputcchar(SCI0, 'l');
fputcchar(SCI0, 'l');
fputcchar(SCI0, 'o');
fputcchar(SCI0, '\n'); /* newline */
fputcchar(SCI0, '\r'); /* carriage return */
```

When you run the program, what do you see on a screen?

The last two characters are special characters which put the text cursor on a new line and then bring the cursor back to the beginning of the line. Printing messages would be a lot easier if we defined a function that called `fputcchar()` for each character in a string such as “hello\n\r”. Fortunately, that function has already been defined:

```
fprintf(SCI0, "hello\n\r");
```

Add the previous line to your program, build and run it. What do you see on a screen? Compare outputs of this and previous exercises. What is the difference between `fputcchar()` and `fprintf()` commands?

Exercise 1: The following program explores the difference between “\n” and “\r”. Run the program and write down the output exactly as you see it. Can you explain the difference between “\n” and “\r”?

```
#include "MRK.h"

int main() {
    fprintf(SCI0, "newline\n");
    fprintf(SCI0, "carriage return\r");
    fprintf(SCI0, "123\n\r");
    fprintf(SCI0, "a carriage return is \r here");
}
```

Write the program output here exactly as you see it:

The `fprintf()` command can do more than print static messages. It can also insert the value of a variable into the output. Enter the following program:

```
#include "MRK.h"

int main() {
    int x,y;
    x = 65;
    y = 43;
    fprintf(SCI0, "x = %d, y = %d\n\r", x, y);
}
```

The first conversion sequence “%d” causes the value of the first argument (`x`) to be converted to the signed decimal number representation. The variable `x` was used in the program as a signed decimal number, so the printed output matches the assignment statement “`x = 65;`”. The second conversion sequence does the same for the variable `y`. But we can interpret a variable using a different representation. Changing the conversion sequences from “%d” to “%c” causes the argument to be interpreted as a character. The output will then be “`x = A, y = +`”. This is an important concept: the 1’s and 0’s stored in memory can represent many different things (signed number, unsigned number, character) depending on how you interpret them. Table 1 lists the conversion sequences that are available. Other formatting options are described in the MRK User Manual, Section 10: Serial Communication Interfaces.

Table 1: `fprintf()` Conversion Sequences

Conversion Sequence	Interpretation
%b	unsigned binary number
%c	single character
%d	signed decimal number
%s	string (array of characters)
%u	unsigned decimal number
%x	unsigned hexadecimal number

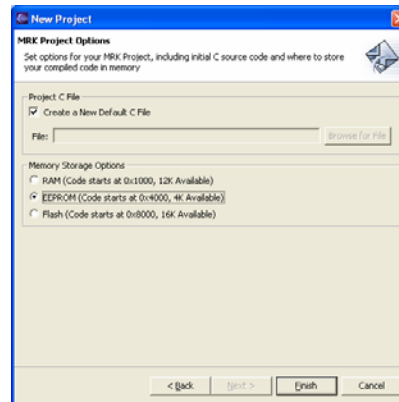
Exercise 2: Complete the following table by using the appropriate conversion sequence. Include your program in the lab report. Explain how your program works.

Notes: 1) Use `int` in your variable declaration

2) A hexadecimal assignment statement looks like `my_var = 0x0041;`

ID	Hexadecimal	Binary	Unsigned Decimal	Signed Decimal	Single Character
1	0x0041				
2	0x0023				
3	0xF2A9				
4	0x80B1				

You will now view on the oscilloscope the transmission of a single character. You will load your program into EEPROM so that you can run the program without being connected to a computer. Create a new MRK project, but choose to store the program in EEPROM:



Enter the following program:

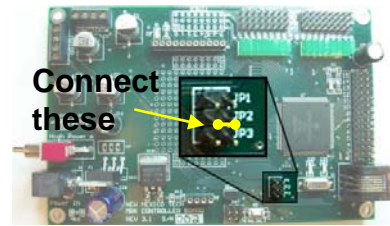
```
#include "MRK.h"

int main() {
    digital_in(INPUT,ALL); /* configure for input */

    while (1) {
        /* if digital_in == 0, write 'a' *
         * if digital_in == 1, write 'b' *
         * 2->'c', 3->'d', and so on */
        fputcchar(SCIO, 'a'+digital_in(IN,ALL));
        /* use delay to show start/stop of transmission */
        delay(3);
    }
}
```

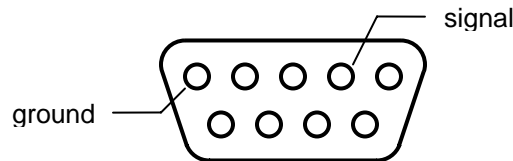
Build and download the program. The program is now in EEPROM. If you set jumper JP2 by connecting a wire between the pins, the program will begin to run automatically after power is connected. Follow these steps carefully:

1. Unplug the power adapter from the MRK board
2. Use a wire to connect jumper JP2.
Ask a TA to verify you have done it correctly.
3. Re-connect the power adapter to the MRK board



With jumper JP2 set, the program in EEPROM will begin running soon after the board has been powered. Connect the power and verify that the character 'a' is being sent to the MRK terminal.

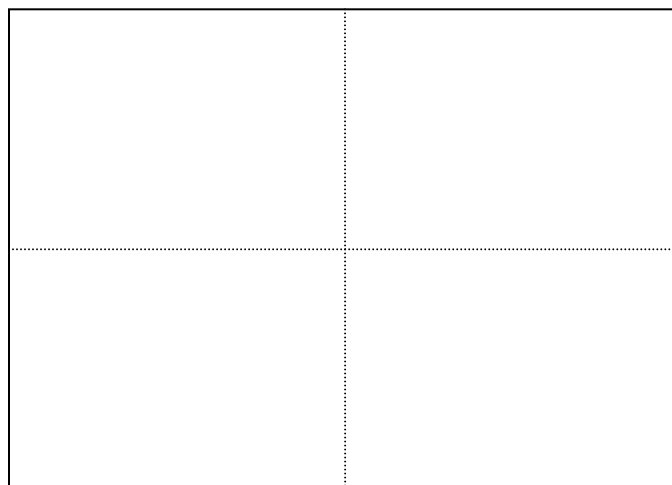
Now you are ready to connect to the oscilloscope. Take the serial cable connector and insert wires into the holes labeled on the following diagram:



Connect the scope probe to these wires. The alligator clip should connect to ground and the spring-loaded tip should connect to signal.

Exercise 3: Connect power to the board. The character 'a' is being sent over the serial link cable. Sketch the signal on the oscilloscope, noting the voltage and time scales. Now, set `digital_in` so that a 'b' or 'c' is being sent (refer to the comments in the program listing). Note how the signal changes.

What voltage represents a logical 1? What voltage represents a logical 0? How much time does the transfer take? How it is related to Baud rate?



4.2 Receiving data

The function `fgetchar(interface)` reads a single character from an SCI interface. The following program demonstrates its usage. Note that the program pauses until it receives a character. We will show in the next section how to respond to keyboard input even when your program is performing other tasks.

```
#include "MRK.h"

int main() {
    char c;
    c = fgetchar(SCI0);
    fprintf(SCI0, "Received the character '%c'\n\r", c);
}
```

Exercise 4: Write a program that will get a character from the keyboard and then display that character and its ASCII code (hint: use different conversion sequences). This part of the program should loop until the last character received was ESC (escape). Note: you will need to look up the ASCII code for ESC in the table at the end of the lab. When your program starts up, print instructions so the user knows what your program does and how to quit. Include your program in the lab report. Explain how it works.

4.3 SCI Interrupt

To use an interrupt, you must define a function containing the statements that should be executed when the interrupt occurs. The function must then be linked to the specific interrupt. The following program illustrates this process.

```
#include "MRK.h"

char c;

void sci0_isr(void) {
    c = fgetchar(SCI0);
}

int main() {
    char count;

    digital_out(OUTPUT, ALL);
    setup_sci(SCI0, INTERRUPT, &sci0_isr);

    c = 0;
    fprintf(SCI0, "Press 'q' to quit.\n\r");
    while (c != 'q') {
        fprintf(SCI0, "c = %c \r", c);
        digital_out(OUT, ALL, count);
        count = count + 1;
        delay(100);
    }
}
```

Conventionally, the function which handles an interrupt is called an Interrupt Service Routine (ISR): `sci0_isr`. This function is linked to the SCI0 interrupt through the `setup_sci()` command.

IMPORTANT WARNINGS: (1) Don't forget the ampersand & when linking the function name. (2) When using the SCI interrupt, you must use the `fgetchar()` function inside the interrupt service routine, or else the program will not work properly.

The declaration for the variable `c` occurs outside of both `main()` and `sci0_isr()` so that both functions can use it. Variables declared inside a function can be seen only by that function.

5 Report and analysis requirements

5.1 Theory

How many types of the serial communication do you know? What are they and how they differ from each other? Which type did you use in laboratory exercises?

What is the ASCII code? Which type of the serial communication it belongs to?

Which bits of the ASCII serial communication format can not be altered while communicating messages? What is the purpose of these bits?

What is the SCI interrupt? Why would one use it? You are designing a mechatronic system with SCI interrupts. List three potential applications of this service routine.

5.2 Results and analysis

Present you results.

List all programs you created during laboratory exercises. Explain how your programs work. Provide line-by-line comments for each program.

Answer all questions in previous sections of these laboratory instructions.

ASCII Character Code Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Source: www.LookupTables.com