Sefa Kayraklık
2015401066

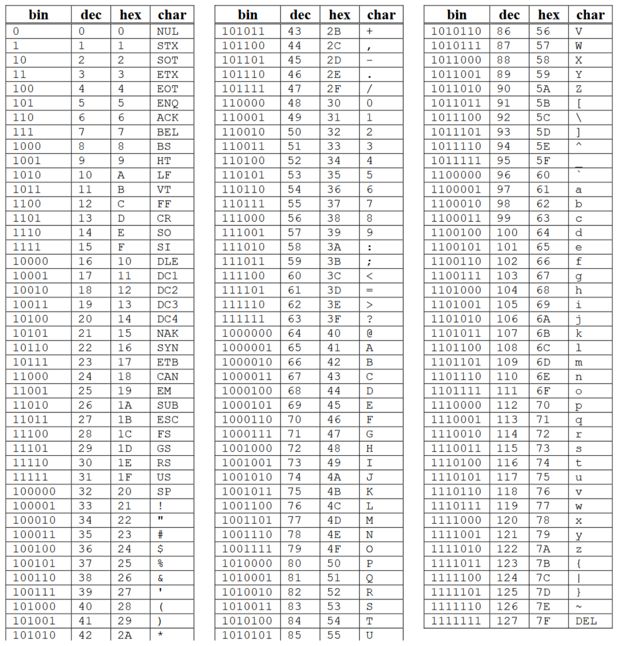
Project 5

EE 479
Fall 2020

I. Introduction

A visible light-based digital communication system can be implemented to transmit and receive digital information. With the means of the arduino setup which was used previous projects, the communication system can be constructed to convey the digital information as ”characters” across 10 centimeters using a white LED and a light-dependent resistor.

In order to convert the characters into bits, the character encoding standard of American Standard Code for Information Interchange (ASCII) is considered. ASCII encodes 128 specified characters, based on English alphabet, into seven-bit representation as shown in the Fig. 1.



bin	dec	hex	char
0	0	0	NUL
1	1	1	STX
10	2	2	SOT
11	3	3	ETX
100	4	4	EOT
101	5	5	ENQ
110	6	6	ACK
111	7	7	BEL
1000	8	8	BS
1001	9	9	HT
1010	10	A	LF
1011	11	B	VT
1100	12	C	FF
1101	13	D	CR
1110	14	E	SO
1111	15	F	SI
10000	16	10	DLE
10001	17	11	DC1
10010	18	12	DC2
10011	19	13	DC3
10100	20	14	DC4
10101	21	15	NAK
10110	22	16	SYN
10111	23	17	ETB
11000	24	18	CAN
11001	25	19	EM
11010	26	1A	SUB
11011	27	1B	ESC
11100	28	1C	FS
11101	29	1D	GS
11110	30	1E	RS
11111	31	1F	US
100000	32	20	SP
100001	33	21	!
100010	34	22	"
100011	35	23	#
100100	36	24	$
100101	37	25	%
100110	38	26	&
100111	39	27	'
101000	40	28	(
101001	41	29	)
101010	42	2A	*
101011	43	2B	+
101100	44	2C	,
101101	45	2D	-
101110	46	2E	.
101111	47	2F	/
110000	48	30	0
110001	49	31	1
110010	50	32	2
110011	51	33	3
110100	52	34	4
110101	53	35	5
110110	54	36	6
110111	55	37	7
111000	56	38	8
111001	57	39	9
111010	58	3A	:
111011	59	3B	;
111100	60	3C	<
111101	61	3D	=
111110	62	3E	>
111111	63	3F	?
1000000	64	40	@
1000001	65	41	A
1000010	66	42	B
1000011	67	43	C
1000100	68	44	D
1000101	69	45	E
1000110	70	46	F
1000111	71	47	G
1001000	72	48	H
1001001	73	49	I
1001010	74	4A	J
1001011	75	4B	K
1001100	76	4C	L
1001101	77	4D	M
1001110	78	4E	N
1001111	79	4F	O
1010000	80	50	P
1010001	81	51	Q
1010010	82	52	R
1010011	83	53	S
1010100	84	54	T
1010101	85	55	U
1010110	86	56	V
1010111	87	57	W
1011000	88	58	X
1011001	89	59	Y
1011010	90	5A	Z
1011011	91	5B	[
1011100	92	5C	\
1011101	93	5D	]
1011110	94	5E	^
1011111	95	5F	_
1100000	96	60	`
1100001	97	61	a
1100010	98	62	b
1100011	99	63	c
1100100	100	64	d
1100101	101	65	e
1100110	102	66	f
1100111	103	67	g
1101000	104	68	h
1101001	105	69	i
1101010	106	6A	j
1101011	107	6B	k
1101100	108	6C	l
1101101	109	6D	m
1101110	110	6E	n
1101111	111	6F	o
1110000	112	70	p
1110001	113	71	q
1110010	114	72	r
1110011	115	73	s
1110100	116	74	t
1110101	117	75	u
1110110	118	76	v
1110111	119	77	w
1111000	120	78	x
1111001	121	79	y
1111010	122	7A	z
1111011	123	7B	{
1111100	124	7C	|
1111101	125	7D	}
1111110	126	7E	~
1111111	127	7F	DEL

Fig. 1. ASCII Table[1]

In the project, the implemented signalling models are **on-off signalling** and **Manchester Coding**.
In the on-off signalling, each bit is indicated by a high voltage if it is 1 or a low voltage if it is 0.
In the Manchester Coding, each bit is indicated by a voltage increase if it is 1 or a voltage drop if it is 0. It means that for the bit of 1, 01 is transmitted and for the bit of 0, 10 is transmitted.
An example of bit sequences which is used the above signalling models is shown in the Fig. 2

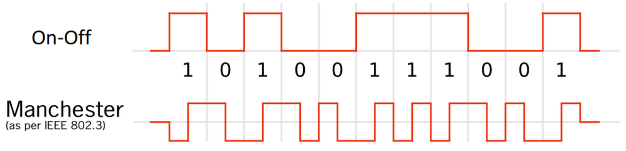


Fig. 2. Example bit sequence, adopted from [2]

II. System Design

As given in the project description, the arduino setup for the visible light-based communication system is constructed and shown in the Fig. 3.

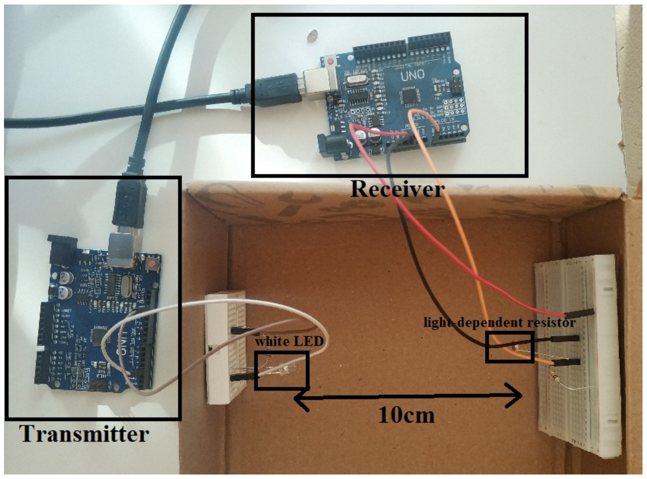


Fig. 3. Arduino setup

A. Transmitting ASCII Codes of Characters:

The following arduino code is used for the transmission of the ASCII codes of characters:

```
//char inChars[12] = "/*12Deneme`"; // $
char inChars[67] = "/*AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz0123456789`";
int delayTime = 3; // specify the waiting time in millisecond
int method = 1; // 1 for on-off, 2 for Manchester
void setup ( ) {
 pinMode(3, OUTPUT); // sets the digital pin 13 as output
}
void loop ( ) {
int charSize = sizeof(inChars);

//delayTime = delayTime + random(-33,33);

/*
 * The chars /* are used for sychronization.
 */
for(int i=0; i<charSize; i++){
  int temp = inChars[i];
  for(int j=0; j<8; j++){
    if (method==1){
      if ((temp >> j) & 1){ // sending on-off signals for corresponding bits
        digitalWrite(3, HIGH);
      }else{
        digitalWrite(3, LOW);
      }
      delay(delayTime);
    }else if (method==2){ // sending Manchester signals for corresponding bits,
      if ((temp >> j) & 1){ // 01 for 1, 10 for 0.
       digitalWrite(3, HIGH);
       delay(delayTime/2);
       digitalWrite(3, LOW);
      }else{
       digitalWrite(3, LOW);
       delay(delayTime/2);
       digitalWrite(3, HIGH);
      }
      delay(delayTime/2);
    }
  }

}
}
```

There exist three inputs which the user should enter: *inChars*, which is the information that is transmitted, *delayTime*, which determines how long a bit is transmitted in millisecond, and *method* is the selection of signalling method.

The transmitted sequence of bits should start with the characters of **/*** which are preambles for synchronization and end with **‘** which is the end-of-transmission character.

B. Receiving ASCII Codes of Characters:

The following arduino code is used for the reception of the ASCII codes of characters:

```
int sensorValue = 0 ;
void setup ( ) {
Serial.begin(115200); } // serial communication
    hizini belirleme
void loop ( ) {
sensorValue = analogRead(A0) ; // A0 portundan
    sample alma
sensorValue = sensorValue / 4 ; // 10 bitlik
    sample iprecision kaybi yaparak 8 bit e
    cevirimis oluyoruz
Serial.write(sensorValue); } // serial port dan
    PC ye yollama
```

C. Decoding Characters from Received ASCII Codes:

The received ASCII codes are processed in MATLAB to decode and to display the sent characters. The MATLAB code consists of one main script and one function, *decode_ASCII*.

The main script calls the function with the user defined parameters of *port name* of the arduino, *buffer size* of the received data, *sampling rate* of the arduino [1], the *duration* of running, the *bitTime* corresponding to delayTime, *threshold* to determine 0s and 1s, and *method* to select signalling model. The following MATLAB code shows the main script.

```
port_name = 'COM3';
buffer_size = 1000;
Fs = 8891;
bitTime = 3; % milisecond
threshold = 40;
method = 'on-off';
%method = 'Manchester';
duration = 10; % how many seconds to run

decode_ASCII(port_name,buffer_size,Fs,bitTime/1e3
    ,threshold,method,duration);
```

[1]It is set to the same value as in the previous projects

The MATLAB function of *decode_ASCII* takes the user defined parameters and runs the amount of seconds which the user gives as *duration*. The ASCII code samples are received as stacks of *buffer size* of points. After receiving the samples as voltages, they are converted to digital samples as using the *threshold* to determine 0s and 1s. The value of the *threshold* is selected by inspecting the received data voltages. After obtaining the digital data, the sent ASCII bits are recovered as counting the number of samples between 0s and 1s. If the number bits are calculated as $round(count/Fs/bitTime)$. With the constructed ASCII bit sequences, the first step is to check the preambles of **/*** to synchronize the receiver. After successful synchronization, the decoded characters are displayed on the command window. And, the display is finished when the end-of-tranmission character of **‘**. The following MATLAB code shows the function.

```
function [] = decode_ASCII(port_name,buffer_size,
    Fs,bitTime,threshold,method,duration)
%decode_ASCII Summary of this function goes here
%   Detailed explanation goes here

priorports=instrfind; % halihazirdaki acik
    portlari bulma
delete(priorports); % bu acik port lari kapama (
    yoksa hata verir)
s = serial(port_name); % bilgisayarinizda hangi
    port olarak define edildiyse,
% o portu girin . . COM1, COM2, vs . .

s.InputBufferSize = buffer_size; % serial
    protokolunden oturu , datayi bloklar halinde
% almaniz gerekiyor. kacar bytelik bloklar
    halinde almak istiyorsaniz, onu girin
set(s, 'BaudRate', 115200) ; % arduino da set
    ettigimiz hiz ile ayni olmali
fopen(s) ; % COM portunu acma

if isequal(method,'Manchester')
  bitTime=bitTime/2;
end

count = 0;
bits = [];
leapData = [];
isSync = 0;
isFinished = 0;
time0 = tic;
while (toc(time0) < duration) && ~isFinished
   data = fread(s);
   transformedData = [leapData; (data<threshold)
       ]; % combining left over data as 0s and 1s

   temp = transformedData(1);
   for i = 1:length(transformedData)
     if transformedData(i)==temp
```

```
         count = count+1;
      else
         numBit = round(count/Fs/bitTime);
         bits = [bits, temp*ones(1,numBit)];
         count = 0;
         temp = ~temp;
      end
      if all(transformedData(i:end)==0)||all(transformedData(i:end)==1)
         leapData = transformedData(i:end);
         break;
      end
   end

   if isequal(method,'Manchester')
      promptBits = bits(2:2:length(bits)); % converting Manchester decoding to ordinary signalling
   elseif isequal(method,'on-off')
      promptBits = bits; % copying the bits to be printed.
   else
      disp('Enter valid method!');
      break;
   end

   if isSync == 0
      for i = 1:length(promptBits)-15
         first = char(bi2de(promptBits(i:i+7)));
         second = char(bi2de(promptBits(i+8:i+15)));
         if isequal([first, second],['/', '*'])
            isSync = 1;
            startIndex = i+16;
         end
      end
   else
      for i = startIndex:8:length(promptBits)-9
         current = char(bi2de(promptBits(i:i+7)));
         if isequal(current,' `') % $
            isFinished = 1;
            fprintf('\nReception is completed. \n');
            break;
         end
         fprintf(current);
         startIndex=startIndex+8;
      end
   end

end
fclose(s); % Serial port u kapatmak
if ~isFinished
   fprintf('\nTime of %d seconds is up. \n', duration);
end
end
```

In the decoding of the Manchester coding, a similar approach in the on-off signalling is used with providing that the bits to prompt are converted to regular bits of on-off signalling by taking the second part of the Manchester coding bits (since 01 is sent to represent 1 and and 10 is sent to represent 0, the every second bit of the sequences gives the actual value).

An example of received samples on-off signalling and Manchester coding with 100 bps data rate is shown in the Fig. 4, respectively.

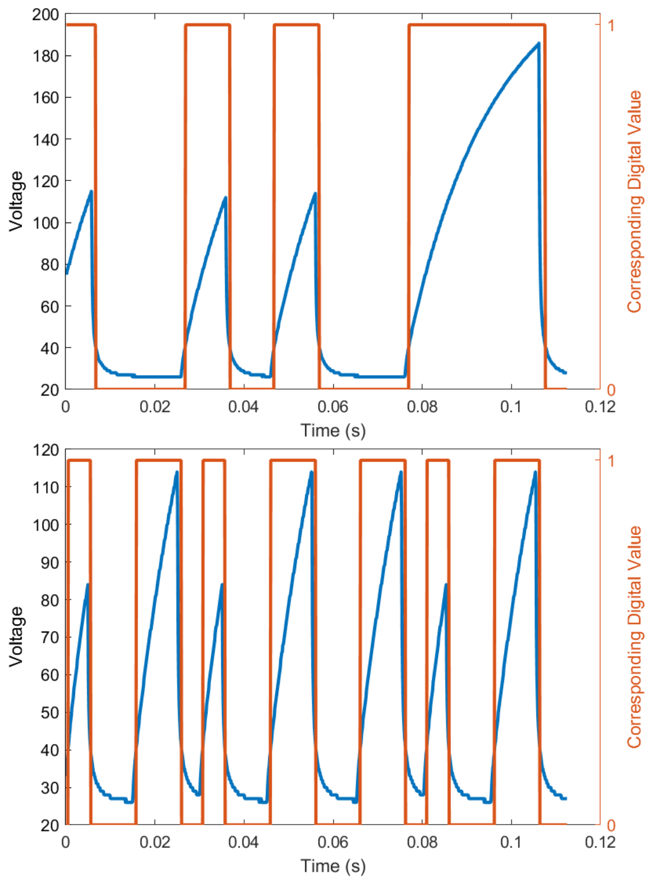


Fig. 4. Examples of received samples of on-off signalling and Manchester coding, respectively

III. Results

A. Data Rate:

The *delayTime* parameter in the transmitting code determines the data rate of the communication system.
So, in the on-off signalling, the decoding algorithm can work properly up to 3ms of bit time which corresponds to the data **333 bits per second**. In the Manchester coding, the decoding algorithm can work properly up to 10ms of bit time which corresponds to the data **100 bits per second**.

B. Unknown Frequency Offset:

In order to observe the robustness of the decoding algorithm, a constant frequency offset is introduced by increasing the *delayTime* while receiver expects to obtain 100ms length of bits.
So, in the on-off signalling, the decoding algorithm can

work properly up to 109ms ($9.17Hz \approx -9\%$) and 90ms (($11.1Hz \approx 11\%$)); therefore, **-9% and 11%** offsets of frequency can be handled by the decoding algorithm. In the Manchester coding, the decoding algorithm can work properly up to 115ms ($8.70Hz \approx -13\%$) and 80ms (($12.5Hz \approx 25\%$)); therefore, **-13% and 25%** offsets of frequency can be handled by the decoding algorithm.

C. Random Clock Delays:

In this section, random clock delays are introduced to test the robustness of the decoding algorithm, while receiver expects to obtain 100ms length of bits.
So, in the on-off signalling, the decoding algorithm can work properly up to random delay of **(-23,23) %**. In the Manchester coding, the decoding algorithm can work properly up to random delay of **(-33,33) %**.

IV. Conclusion

The following table illustrates the above results of the tests.

TABLE I
Comparison of the on-off signalling and Manchester Coding

	On-Off	Manchester
Data Rate [bps]	333	100
Frequency Offset [%]	-9 +11	-13 +25
Random Clock Delay [%]	(-23,23)	(-33,33)

It can be seen from the table that the on-off signalling works faster but less robust than the Manchester coding. Hence, if the transmitter is very much stable, the on-off signalling would be wise to use; whereas, in the real life, the transmitter is not so much stable, so using Manchester coding is preferable since it gives higher performance under unstable conditions, which means it is more robust.

References

[1] Binary/Decimal/Hexadecimal/ASCII Character Conversion Chart. Retrieved 14 January 2021, from https://student.cs.uwaterloo.ca/~cs241/ConversionChart.pdf
[2] Retrieved 14 January 2021, from https://en.wikipedia.org/wiki/Manchester_code

Appendix

The MATLAB and arduino codes that are used in this project is in the *EE479Project5(Sefa Kayraklık).zip* file. The content of file is following m, ino files and the reports:

- Main.m and decode_ASCII.m are for the MATLAB part
- sketch_nov11a.ino and generate_code.ino are for the arduino part
- EE479Project5.pdf for the reports