Analog - Simple PC Oscilloscope (Using TX8 and SAR6)

AN2106

Author: Mehmet Zeki SONMEZ
Associated Project: Yes
Associated Part Family: CY8C25xxx, CY8C26xxx
GET FREE SAMPLES HERE
Software Version: NA
Associated Application Notes: None

Application Note Abstract

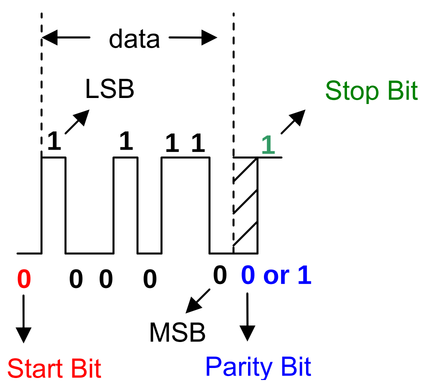
This project illustrates how to convert analog information to digital information and then send it to the PC via RS232 using an 8-bit serial transmitter. Software is also included in the project file for data logging and drawing the incoming signal shape. Before transmitting data to a PC we will use the PSoC® Pup hardware to understand the 8-bit serial transmitter (TX8) operations. For more details on the SAR6 User Module, see AN2093, Keypad Scan using ADC (SAR6).

Introduction

For asynchronous serial communication, the data stream is composed of a start bit, actual data, parity bit (optional), and a stop bit.

Examine Figure 1 for the data 01101001 (69h) and one parity bit:

Figure 1



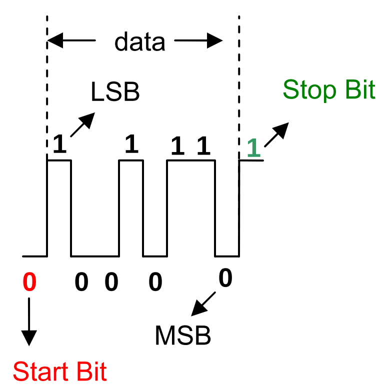
Before sending the actual data, a start bit is sent to the receiver. This informs the receiver that the next 8 bits will be the actual data. Because the receiver and transmitter work with the same baud rates, the receiver will know how much time passes before the next bit comes. For example, if the baud rate is 9600 bits per second, the receiver will read the next bit every 1/9600 second later.

Once the receiver has the 8-bit data (69h, in our example), the next bit will be the parity bit. The receiver will use the parity bit if incoming data is correct or not.

The final bit will be the stop bit, which informs the receiver that the incoming byte has finished. The receiver will not read data until a new start bit is initiated.

Examine Figure 2 for the data 01101001 (69h) but this time no parity bit is used:

Figure 2



8-Bit Serial Transmitter TX8

TX8 is an 8-bit serial transmitter, which is convenient for RS232 data format. Load the data to the Accumulator A and send it by using one command! Following are detailed descriptions of how you can use it.

Determining Baud Rate of TX8

The clock of TX8 must be eight times the baud rate. For example, if the baud rate of serial transmission is 9600 bps, then the input clock of the TX8 has to be 8x9600 = 76800 Hz. Using the Counter or PWM User Module as a divider, we can get this frequency:

Input Clock of TX8 = 8 x Baud Rate

Project 1: PSoC Pup Project

For better understanding and visual perception we will use the PSoC Pup hardware and adjust the baud rate as minimum so we can see with our eyes what actually happens. This project can be found in `prj1_tx8_pup` directory in the project files of this Application Note.

For this particular project, we will use a 16-bit PWM and a TX8 User Module.

Adjust Global Parameters

24V1 = 16

24V2 = 16

24V1 is now 24 MHz / 16 → 1.5 MHz

24V2 is now 1.5 MHz / 16 → 93.75 kHz

Adjust PWM16 Parameters:

Place PWM16 at DBA02 and DBA03 blocks.

- Clock = 24V2
- Enable = High
- Period = 65535
- PulseWidth = 32768
- CompareType = Less than or equal
- InterruptType= Terminal Count
- Output = Global_OUT_0

Output of PWM is now:

- 24V2 / 65535
- 93.75 kHz/ 65536 → 1.4305 Hz

This will be the input for TX8. Remember that:

- Input Clock of TX8 = 8 x Baud Rate.
- 1.4305 Hz = 8 x Baud Rate.
- Baud Rate = 0.1788 bps! (This is a very slow transfer rate but our aim is watching bits on the LEDs.)

Adjust TX8 Parameters

Place TX8 at DCA04.

- Clock = DBA03
- Output = Global_Out_4
- Parity = None

Adjust Pin Configurations

PORT2_0 = Global_Out_0 STRONG

PORT2_4 = Global_Out_4 STRONG

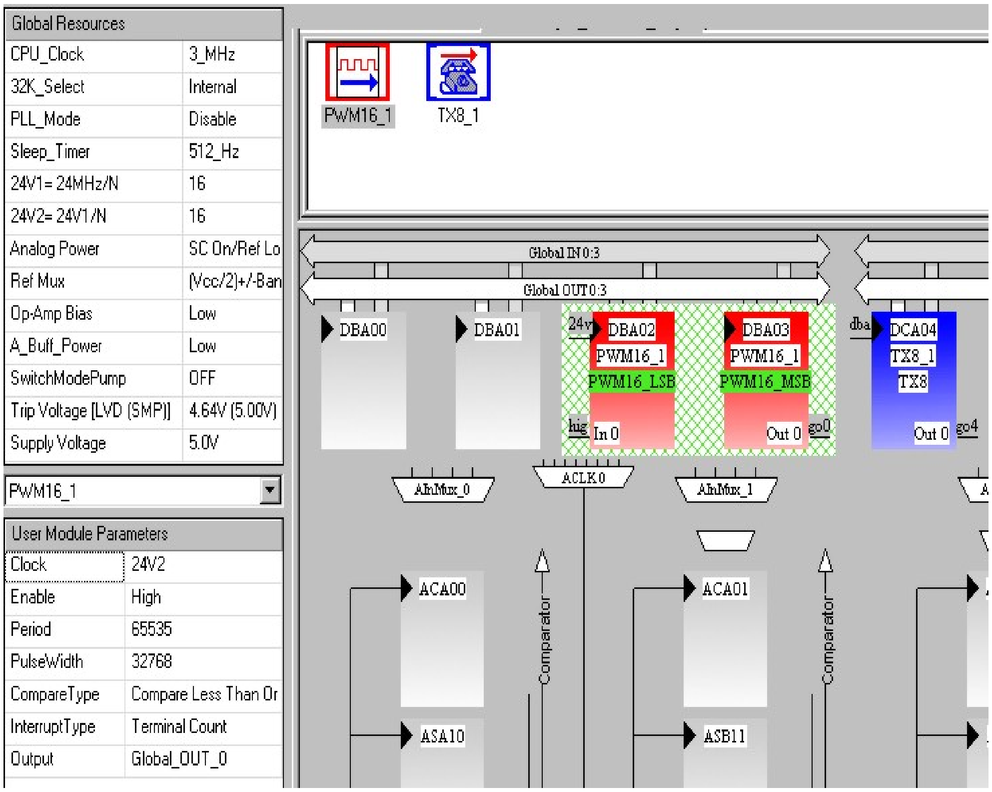
We will use Port2_0 for watching the clock on LED 10 of the PSoC Pup and we will use Port2_4 for watching serial transmission on LED 6. See AN2011, PSoC_Pup Example Projects, for PSoC Pup hardware schematic.

For graphical view of the User Modules for this project see Figure 3.

Here is the *main.asm* file:

```
include "m8c.inc"
include "TX8_1.inc"

export _main

_main:
//Place a breakpoint here
        call  TX8_1_Start
        call  PWM16_1_Start

        mov   A,47h
        call  TX8_1_SendData
        ret
```

Figure 3



Now place a breakpoint to the first line after `_main` and run your project step by step. While you pass call PWM16_1_Start, the LED on the Pup will flash (that is the clock of the TX8). Now run again and observe the LEDs for the transmitting data (47h).

Note LED 6 (output of TX) changes in every eight clocks.

TX output LED will be ON in the first 16-clock period (LED 10 will flash 16 times). Then logic 0 comes (start bit), after that the data and finally the stop bit will come. Note that the first bit of the data sent is the least significant bit. Observe the diagram in Figure 4.

Figure 4

nth **Clock**	**Led 6 TX Output**		
0	1 -		
8	1 -		
16	0 – Start Bit		
24	1 – Bit 0 (LSB)	7h	**47h**
32	1 – Bit 1		
40	1 – Bit 2		
48	0 – Bit 3		
56	0 – Bit 4	4h	
64	0 – Bit 5		
72	1 – Bit 6		
80	0 – Bit 7(MSB)		
88	1 – Stop Bit		

Project 2: Sending Data to PC

We will use the *datalogger* software that is included in the project file to see the data received from PSoC.

Before we start our project we have to determine the rules for serial communication. We will use the following rules:

- Baud Rate: 300 bps
- 8 Data Bit
- 1 Stop Bit
- No Parity

Both the transmitter (PSoC) and the receiver (PC) must obey these rules. Otherwise, data cannot be transmitted correctly.

For a 300 bps baud rate we need 8x300 bps = 2400 Hz for input clock of TX8.

This time an 8-bit PWM is enough for getting 2400 Hz. We will divide 93750 kHz by 39 to get 2400 Hz. Placement of the User Modules is shown in Figure 5. Here are the parameters:

Adjust Global Parameters

24V1 = 16

24V2 = 16

24V1 is now 24 MHz / 16 → 1.5 MHz

24V2 is now 1.5 MHz / 16 → 93.75 kHz

Adjust PWM8 Parameters

Place PWM8 at DBA03.

- Clock = 24V2
- Enable = High
- Period = 38
- PulseWidth = 19
- CompareType = Less than or equal
- InterruptType = Terminal Count
- Output = None

Output of PWM is now:

- 24V2 / 39
- 93.75 kHz / 39 → 2403.8 Hz

This will be the input of TX8. Remember that:

- Input Clock of TX8 = 8 x Baud Rate.
- 2403.8 Hz = 8 x Baud Rate.
- Baud Rate = 300 bps.

Adjust TX8 Parameters

Place TX8 at DCA04.

- Clock = DBA03
- Output = Global_Out_4
- Parity = None

Adjust Pin Configurations

PORT1_4 = Global_Out_4 STRONG

(Pin17 for 26443)

Now we will send "I LOVE PSoC" to the PC. ASCII Character Map can be found in the Appendix.

I → 49h

Space → A0h

L → 4Ch

O → 4Fh

V → 56h

E → 45h

Space → A0h

P → 50h

S → 53h

o → 6Fh

C → 43h

Space → A0h

A table can be formed in order to simplify the assembler code.

Mytable:

db 49h , A0h , 4Ch , 4Fh , 56h , 45h , A0h , 50h , 53h , 6Fh , 43h , A0h

We use the following code to check if the byte is sent:

```
call  TX8_1_SendData       ;Send A to
receiver
TXnotready:
call bTX8_1_ReadTxStatus
and  A, TX8_TX_COMPLETE
jz  TXnotready
```

When a series of bytes is sending to the receiver it is necessary to check if the current byte has been sent to the receiver. In other words, if TX8 is completed the transmission for the current byte. Otherwise, if you do not check for the completion, then TX8 will not work correctly.

You cannot do this:

```
mov A,10h
call  TX8_1_SendData       ;Send A to
receiver
mov A,20h
call  TX8_1_SendData       ;Send A to
receiver
```

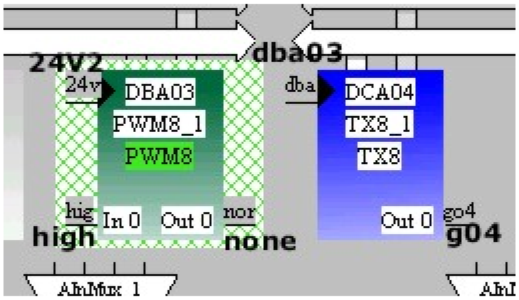
20h will probably not transmit so we have to add a check progress to code.

You can do this:

```
mov  A,10h
call   TX8_1_SendData      ;Send A to
receiver
TXnotready:
call   bTX8_1_ReadTxStatus
and  A,TX8_TX_COMPLETE
jz     TXnotready
mov  A,20h
call   TX8_1_SendData      ;Send A to
receiver
```

```
include "tx8_1.inc"
include "pwm8_1.inc"
include "m8c.inc"
export _main
_main:

temp:  equ 00h            ;[temp] or [00] will store number n(nth element of the table)

call   PWM8_1_Start
call   TX8_1_Start

send_table:

mov [temp],0        ;First element of the table will be loaded in to accumulator A

send_next_byte:

mov A,[00]          ;Load the element number of the table to A
index MyTable       ;Retrieve the relevant element from the table to accumulator A
call  TX8_1_SendData       ;Send the accumulator A value to PC

TXnotready:         ;Check if the byte is sent to the receiver?
call bTX8_1_ReadTxStatus
and  A, TX8_TX_COMPLETE
jz  TXnotready             ;No it is still sending , check again if progress is complete

;Data(1 Byte)is sent now. Check if we are at the end of the Table?
;If no Retrieve the next element of the Mytable and send it to PC

cmp [00],11         ;Are we at the end of the Mytable?
jz  send_table             ;Yes send the whole table to PC again

;No there are still some elements which should be sent
;Then send the next element to PC

inc    [00]
jmp send_next_byte


MyTable:
     db 49h,A0h,4Ch,4Fh,56h,45h,A0h,50h,53h,6Fh,43h,A0h     ;data sent to TX
;           I     spc        L     O    V     E    spc    P      S    o     C    space
```

Figure 5

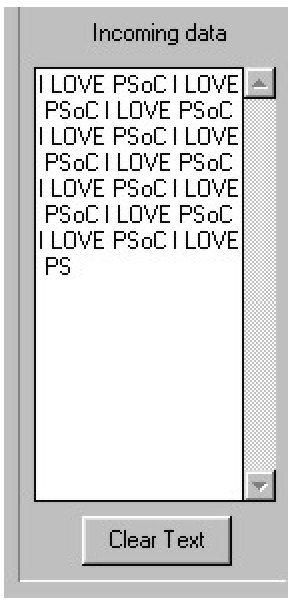


Now configure your hardware according to Schematic 1, which is at the end of this Application Note. Then run your project at the same time with datalogger software.

Select which COM port you use and set the baud rate as 300 bps. Then click on the 'Open Port ' button.

You will see the incoming data from PSoC in the box in the software as shown in Figure 6:

Figure 6



Note When you click on 'Open Port', a file is created at `C:\psocdata.txt`. When you click on 'Close Port' the content of the memo box is stored to `C:\psocdata.txt`. You can use this file in your particular projects.

Project 3: Simple PC Oscilloscope

Implementing the hardware of this PC-based oscilloscope is very easy. All we do is sample the analog signal and convert it to digital, finally sending the digital code to the PC. The software *datalogger* gets the digital code and draws the approximate incoming signal shape. Because *datalogger software* stores the incoming data in file `C:\psocdata.txt` you can use this file to design your own software.

The Idea

SAR6 (ADC) gets the analog signal from Port_0_2 by using a buffer (PGA, gain=1) and then converts the analog signal to digital code. These digital code bytes are stored into the RAM area of the PSoC and placed in addresses from 0x02 to 0xEF.

Once the RAM area is filled (when the last data is stored to EFh), all stored bytes are sent to the PC at 111111 bps. Because this takes some time, PSoC cannot perform the analog to digital conversion and will miss some signals when it is sending data to the PC. Thus, the drawn signal in the *datalogger* is intermittent. By using an appropriate algorithm on the PC side, you can improve the signal shape (for periodic signals) and add zoom-in, zoom-out properties.

Here are the parameters for this project:

Adjust Global Parameters

24V1 = 3

24V2 = 9

Ref Mux = $(V_{cc}/2)\pm(V_{cc}/2)$

All other global parameters are default values. You can check them in the `prj_3_pc_osc` files.

24V1 is now 24 MHz / 3 → 8 MHz

24V2 is now 8 MHz / 9 → 888.888 kHz

Adjust TX8 Parameters

Place TX8 at DCA04.

- Clock = 24V2
- Output = Global_out_4
- Parity = None
- InterruptAPI = Disable

888.888 kHz / 8 = 111111 bps (Baud Rate)

Adjust PGA Parameters

Place PGA at ACA03.

- Gain = 1
- Input = AnalogColumn_InputMUX_3
  Select Port_0_2 in AlnMux_3
- Reference = AGND
- AnalogBus = Disable

Adjust SAR6 Parameters

Place SAR6 at ASB13.

- SignalSource= ACA03
- ClockPhase = Normal

Select 24V2 for AnalogColumnClock_3

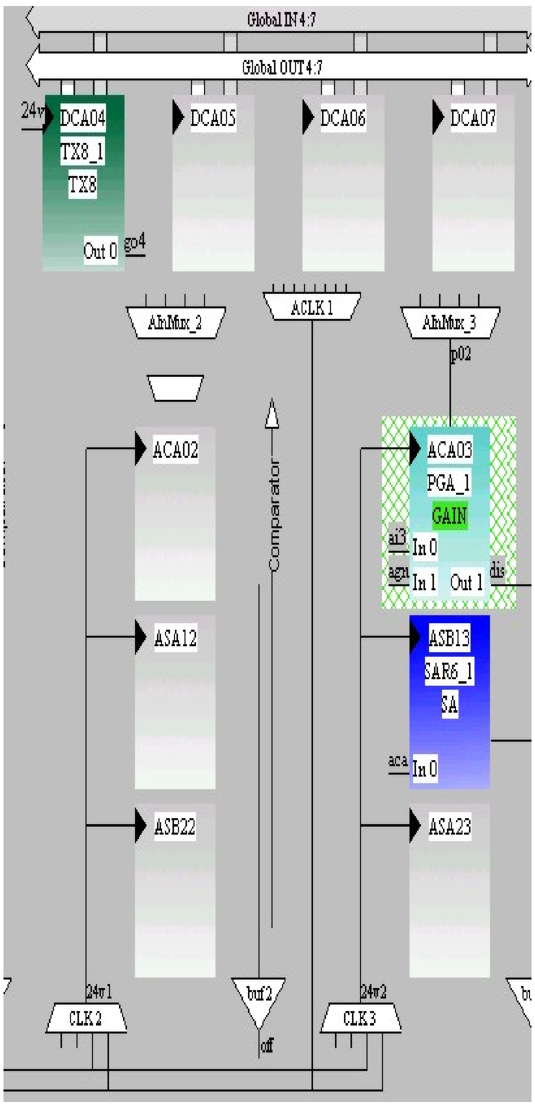
Pin Configuration

Configure Port_0_2 as Analoginput HigZ

Configure Port_1_4 as Global_out_4 Strong

See Figure 7 for User Module placement view:

Figure 7



We send a sign, which is the ASCII character 2 (decimal code of '2 'is 50), after each array of bytes (data in RAM [0x02-0xEF]) is sent to the PC. So the PC knows that there will be a delay for the next-coming bytes when it receives the sign. Because of this, the graph drawn is intermittent.

Another point is that the SAR6 produce values between [-31 and 31] and the ASCII characters start from 33 (decimal). *Datalogger* software cannot understand if PSoC is sending a value between 0 and 31 because the mentioned range does not include ASCII characters.

Thus, a simple code is used to add 33 to the converted digital code, providing that it is in the range of 0-31.

For example, if the amplitude for the analog signal is 2.5 V, which is equal to the reference, the SAR6 will produce 00h. This 00h will be added with 33. Thus, 33 will represent 00h and 34 will represent 01h, 35 will represent 02h, and so on. If the digital code that is produced is between -31 and -1 or between E1h and FFh, it will not be converted to another code because that code represents the ASCII characters. ASCII characters run from 33 to 255.

You can find ASCII characters and their code in the Appendix.

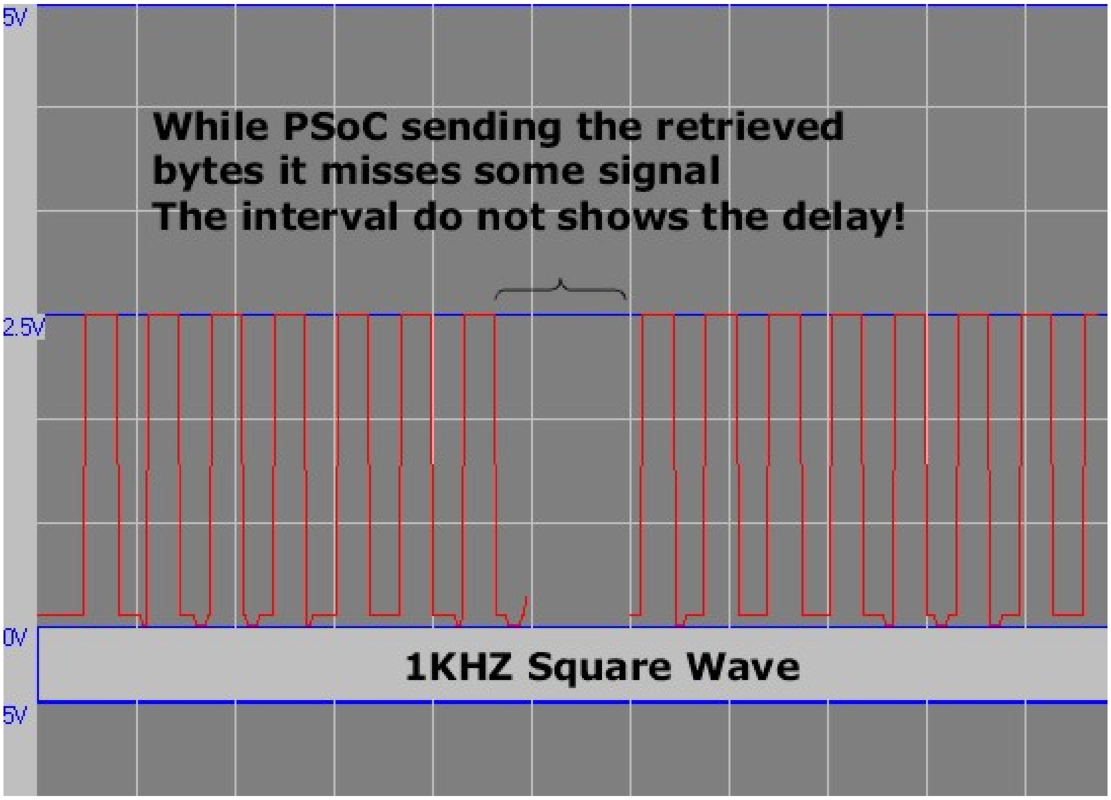
The detailed description can be found in the assembler code on the next page.

After configuring the hardware, as seen in Schematic 2 at the end of this Application Note, run the *datalogger* software and run your project. Configure the COM port and adjust the baud rate to 111111 bps. Then click on 'Open Port' to retrieve data. Note that the analog signal is measured by port_0_2 and must be between **0 V-5 V**. The best graph can be obtained near 1 kHz frequencies. See Figure 8 for a screenshot of the *datalogger* software measuring a 1 kHz square wave.

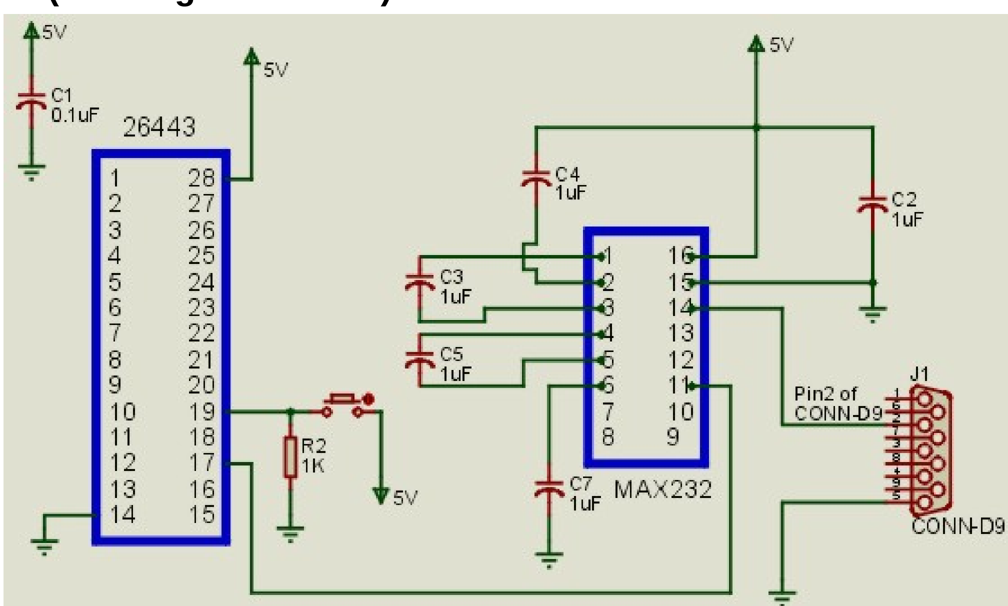
As a reminder, the captured data with ASCII character '2's decimal value = 50 (showing new array of bytes) is stored to `C:\psocdata.txt` file. You can write your own software or use this software for your serial communication projects.

```
include "SAR6_1.inc"
include "PGA_1.inc"
include "TX8_1.inc"
include "m8c.inc"
export _main
area bss(RAM)
    buffer:   BLK  F0h     ;Captured data is stored in this RAM block
area text(ROM,REL)
_main:
temp:  equ 00h
   mov A,PGA_1_HIGHPOWER
   call  PGA_1_Start
   call  TX8_1_Start
   mov A,SAR6_1_HIGHPOWER
   call SAR6_1_Start
sampleagain:
       mov X,01h     ;Ram[01] will hold the 50-->ASCII character '2'. This is a sign that
       mov [x],50    ;informs the PC a new array of bytes is coming. PC will draw the graph
                     ;after a little interval.
       inc x         ;Ready for writing data to RAM[02]
   store:
   call SAR6_1_GetSample   ;This will get the output byte of SAR6 to A
   mov [temp],A            ;Retrieved data is stored to temporary RAM area
   sub A,32 ;        ;Is the retrieved data between 0 and 31?
   jnc goon          ;No it is not, then this is an ASCII character, no problem.
   add [temp],33           ;Yes, the retrieved data is between 0 and 31. Then add 33 to this
data
                     ;so we can make it an ASCII character

goon:
  mov A,[temp]             ;Captured data is loaded to A again.
  mov [X],A          ;Store the retrieved byte to the RAM location
  inc   X            ;Ready for next byte
  mov A,X            ;Load Index register to A
  cmp A,F0h          ;Check if the RAM is full of retrieved data?
  jnz  store         ;No we have empty RAM areas which should be filled with retrieved bytes

;RAM area[02-EF] is filled with digital codes of analog signal
;Now it is time to send these bytes to PC
;PSoC will miss some of data while it is sending the stored bytes

  mov X,00h          ;Index register shows RAM[00]
send:
  inc X                    ;Go to the next RAM address. Note that RAM[01] includes a sign
                     ;which means it is the beginning of a new array of bytes.
 mov A,X
 cmp A,F0h           ;Check if PSoC has completed the process(serial transmission of bytes)
 jz sampleagain            ;If yes, then sample and store the next bytes for incoming signal

mov A,[X]            ;If the process is not finished yet then load the byte which will be sent
call  TX8_1_SendData       ;send the byte to receiver

isitready:           ;Check if the serial transmission of the current byte is finished?
 call bTX8_1_ReadTxStatus
 and  A, TX8_TX_COMPLETE
 jz  isitready             ;If PSoC still sending the current byte then wait until it is
ready for next byte

jmp     send         ;Send the next byte to the PC

   ret
```

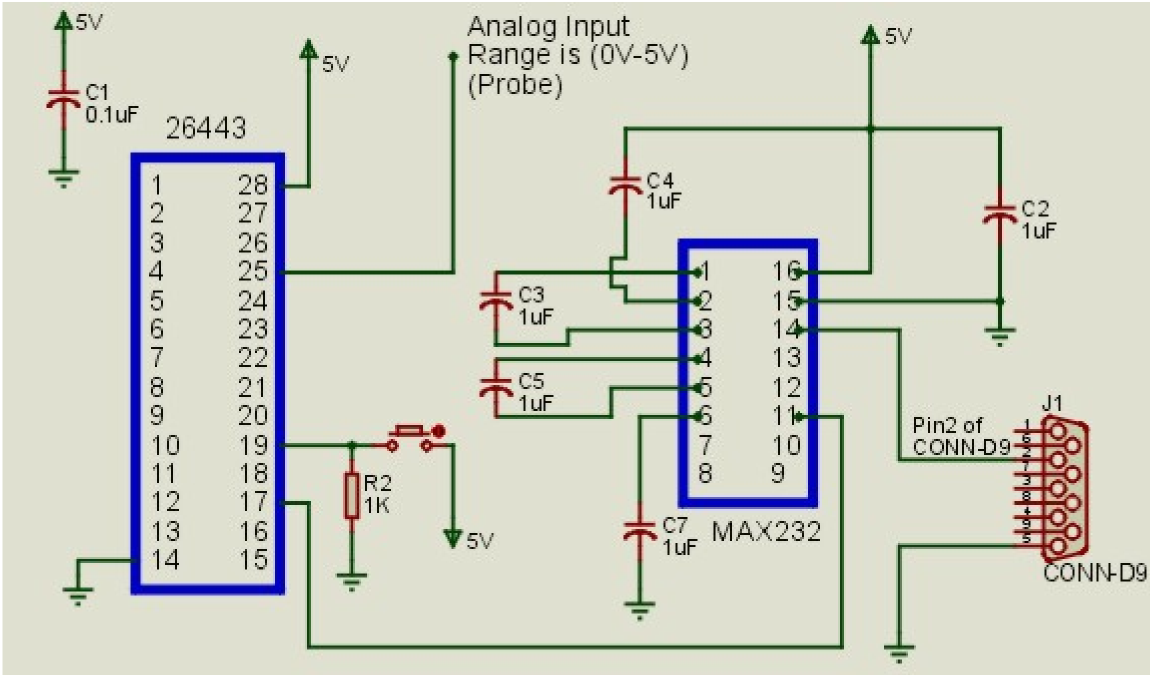
Figure 8



Schematic 1 (Sending Data to PC)



Schematic 2 (PC Oscilloscope)



The schematics are almost the same. The only difference is in Schematic 2 (PC Oscilloscope) pin 25 (port_0_2) is configured as analog input so we can capture the analog signals.

Appendix. ASCII Character Map

Decimal	Character	Binary	Hex		Decimal	Character	Binary	Hex
33	!	100001	21		91	[	1011011	5B
34	"	100010	22		92	\	1011100	5C
35	#	100011	23		93	]	1011101	5D
36	$	100100	24		94	^	1011110	5E
37	%	100101	25		95	_	1011111	5F
38	&	100110	26		96	`	1100000	60
39	'	100111	27		97	a	1100001	61
40	(	101000	28		98	b	1100010	62
41	)	101001	29		99	c	1100011	63
42	*	101010	2A		100	d	1100100	64
43	+	101011	2B		101	e	1100101	65
44	,	101100	2C		102	f	1100110	66
45	-	101101	2D		103	g	1100111	67
46	.	101110	2E		104	h	1101000	68
47	/	101111	2F		105	i	1101001	69
48	0	110000	30		106	j	1101010	6A
49	1	110001	31		107	k	1101011	6B
50	2	110010	32		108	l	1101100	6C
51	3	110011	33		109	m	1101101	6D
52	4	110100	34		110	n	1101110	6E
53	5	110101	35		111	o	1101111	6F
54	6	110110	36		112	p	1110000	70
55	7	110111	37		113	q	1110001	71
56	8	111000	38		114	r	1110010	72
57	9	111001	39		115	s	1110011	73
58	:	111010	3A		116	t	1110100	74
59	;	111011	3B		117	u	1110101	75
60	<	111100	3C		118	v	1110110	76
61	=	111101	3D		119	w	1110111	77
62	>	111110	3E		120	x	1111000	78
63	?	111111	3F		121	y	1111001	79
64	@	1000000	40		122	z	1111010	7A
65	A	1000001	41		123	{	1111011	7B
66	B	1000010	42		124	|	1111100	7C
67	C	1000011	43		125	}	1111101	7D
68	D	1000100	44		126	~	1111110	7E
69	E	1000101	45		127	•	1111111	7F
70	F	1000110	46		128	•	10000000	80
71	G	1000111	47		129	•	10000001	81
72	H	1001000	48		130	•	10000010	82
73	I	1001001	49		131	•	10000011	83
74	J	1001010	4A		132	•	10000100	84
75	K	1001011	4B		133	•	10000101	85
76	L	1001100	4C		134	•	10000110	86
77	M	1001101	4D		135	•	10000111	87
78	N	1001110	4E		136	•	10001000	88
79	O	1001111	4F		137	•	10001001	89
80	P	1010000	50		138	•	10001010	8A
81	Q	1010001	51		139	•	10001011	8B
82	R	1010010	52		140	•	10001100	8C
83	S	1010011	53		141	•	10001101	8D
84	T	1010100	54		142	•	10001110	8E
85	U	1010101	55		143	•	10001111	8F
86	V	1010110	56		144	•	10010000	90
87	W	1010111	57		145	•	10010001	91
88	X	1011000	58		146	•	10010010	92
89	Y	1011001	59		147	•	10010011	93
90	Z	1011010	5A		148	•	10010100	94

Decimal	Character	Binary	Hex		Decimal	Character	Binary	Hex
149	•	10010101	95		203	Ë	11001011	CB
150	•	10010110	96		204	Ì	11001100	CC
151	•	10010111	97		205	Í	11001101	CD
152	•	10011000	98		206	Î	11001110	CE
153	•	10011001	99		207	Ï	11001111	CF
154	•	10011010	9A		208	Ğ	11010000	D0
155	•	10011011	9B		209	Ñ	11010001	D1
156	•	10011100	9C		210	Ò	11010010	D2
157	•	10011101	9D		211	Ó	11010011	D3
158	•	10011110	9E		212	Ô	11010100	D4
159	•	10011111	9F		213	Õ	11010101	D5
160		10100000	A0		214	Ö	11010110	D6
161	¡	10100001	A1		215	×	11010111	D7
162	¢	10100010	A2		216	Ø	11011000	D8
163	£	10100011	A3		217	Ù	11011001	D9
164	¤	10100100	A4		218	Ú	11011010	DA
165	¥	10100101	A5		219	Û	11011011	DB
166	¦	10100110	A6		220	Ü	11011100	DC
167	§	10100111	A7		221	İ	11011101	DD
168	¨	10101000	A8		222	Ş	11011110	DE
169	©	10101001	A9		223	ß	11011111	DF
170	ª	10101010	AA		224	à	11100000	E0
171	«	10101011	AB		225	á	11100001	E1
172	¬	10101100	AC		226	â	11100010	E2
173	-	10101101	AD		227	ã	11100011	E3
174	®	10101110	AE		228	ä	11100100	E4
175	¯	10101111	AF		229	å	11100101	E5
176	°	10110000	B0		230	æ	11100110	E6
177	±	10110001	B1		231	ç	11100111	E7
178	²	10110010	B2		232	è	11101000	E8
179	³	10110011	B3		233	é	11101001	E9
180	´	10110100	B4		234	ê	11101010	EA
181	µ	10110101	B5		235	ë	11101011	EB
182	¶	10110110	B6		236	ì	11101100	EC
183	•	10110111	B7		237	í	11101101	ED
184	¸	10111000	B8		238	î	11101110	EE
185	¹	10111001	B9		239	ï	11101111	EF
186	º	10111010	BA		240	ğ	11110000	F0
187	»	10111011	BB		241	ñ	11110001	F1
188	¼	10111100	BC		242	ò	11110010	F2
189	½	10111101	BD		243	ó	11110011	F3
190	¾	10111110	BE		244	ô	11110100	F4
191	¿	10111111	BF		245	õ	11110101	F5
192	À	11000000	C0		246	ö	11110110	F6
193	Á	11000001	C1		247	÷	11110111	F7
194	Â	11000010	C2		248	ø	11111000	F8
195	Ã	11000011	C3		249	ù	11111001	F9
196	Ä	11000100	C4		250	ú	11111010	FA
197	Å	11000101	C5		251	û	11111011	FB
198	Æ	11000110	C6		252	ü	11111100	FC
199	Ç	11000111	C7		253	ı	11111101	FD
200	È	11001000	C8		254	ş	11111110	FE
201	É	11001001	C9		255	ÿ	11111111	FF

About the Author

Name: Mehmet Zeki SONMEZ

Title: Yeditepe University: Electrical & Electronics Engineering Student & Student Assistant (last year).

Background: M. Zeki Sonmez is interested in design of both analog and digital (including microcontrollers) circuits. In the near future his area of interest will be RF Identification.

Contact: zeki@sonmezticaret.com or mzsonmez@hotpop.com http://electronicsclub.cjb.net

In March of 2007, Cypress recataloged all of its Application Notes using a new documentation number and revision code. This new documentation number and revision code (001-xxxxx, beginning with rev. **), located in the footer of the document, will be used in all subsequent revisions.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com/