8-CHANNEL 'NANO-CTA' ARRAY THERMAL ANEMOMETRY SYSTEM

User Manual

WARNING

Read this document before using the product.

This probe is an experimental prototype, for measurement purposes only.

This system is not certified for use on aircraft.

Contents

Version Control

Version	Date	Summary of changes
1.0	09-2019	New document
1.1	10-2019	Updated checksum, power consumption and handling/ installation instructions. Addition of guidance on output variable X and CRC-16-CCITT implementation.

1 INTRODUCTION

Principle of operation

This system consists of up to 8 miniaturized, surface-mountable flexible thermal anemometry sensing elements (with local signal conditioning) and a data acquisition unit which can stream data to a control system, or to a computer for acquisition.

The sensing elements are a proprietary CMOS-based technology, and are able to resolve ultralow flow speeds, with temperature compensation implemented in hardware. The system will provide the bridge voltage and fluid temperature. Note that in-situ velocity calibration is usually required. Calibration drift will be within ~2% over long periods of use or storage.

System description

Miniaturized multichannel low-velocity thermal anemometry system.

System components

(X) Sensing units (**Error! Reference source not found.**Figure 2)	Each sensing unit has one temperature and one velocity sensor with analogue-balance temperature compensation system.
1x Data acquisition unit (Figure 1)	This supports up to 8 simultaneous channels and provides data streaming via USB connection.
(X) Sensor cable assemblies	4-way, 600 mm cable assemblies for connecting sensing elements to data acquisition unit
1x USB cable	A low-profile USB micro->A connector is bundled with the system.

Please ensure that all the system components listed above have been supplied, and that there is no apparent damage from shipping.

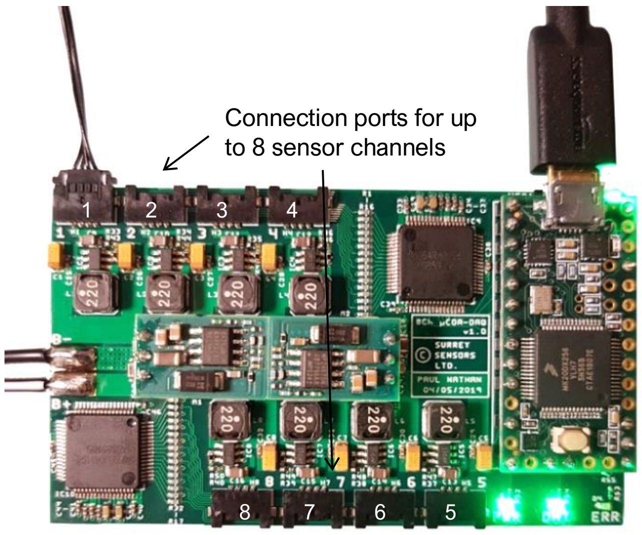


Figure 1: Eight channel data acquisition unit

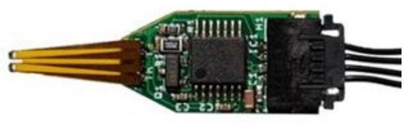


Figure 2: Individual sensing unit

System requirements

To interface with a computer, the probe system requires 64-bit Windows 7 (or newer) operating system (not included). Note that computer interface is not needed for stand-alone streaming operation. The probe has been pre-loaded with firmware; the computer software drivers and data logging software will be provided electronically.

2 DETAILED SPECIFICATION

Velocity range	< 10 mm/s to 120 m/s (custom extended range available)
Uncertainty	± 1 % relative
Compensated temperature range	0° to 70° C ambient for dry air
Calibration drift	< 2 % over long periods of use or storage
Storage temperature range	-40° to +85° C
Maximum relative humidity	95 % (non-condensing)
Communications interface	Data streaming via USB2.0 or TTL
Power	7 – 36 VDC supply required, min. 3W, max. 6W plus USB typical 300 mW
Data acquisition rate	Up to 400 Hz simultaneous
Digital resolution	16-bit
System requirements[1]	64-bit Windows 7 or later
Physical dimensions	Sensor package approx. 10 mm x 20 mm x 2 mm Data acquisition unit approx. 45 x 77 mm x 6 mm

[1]Note that computer interface is not needed for stand-alone streaming operation

3 SERIAL COMMUNICATIONS

When the system is powered on, it will undergo a brief system diagnostic test; if the test is passed, then a green LED will illuminate at the rear of the probe near the cable connections.

Commands
Each data packet consists of 131 bytes, beginning with an unsigned-integer frame character ("#") and terminating with a checksum (both inclusive); the UART configuration is the typical 8-N-1. The data order is shown below. The variables given in the table are:

- **X –** Temperature compensated non-dimensional output that is a function only of Reynolds number. i.e. X = f(Re). Further information on use of this variable is appended to the end of this document.
- **V_s –** Sensor element supply raw voltage (un-corrected). Useful for diagnostic purposes as well as to check for saturation, or for user's own temperature correction.
- **T_w –** Hot sensor temperature.
- **T_a –** Cold sensor temperature, which is approximately equal to the ambient temperature (used for the temperature corrections).

byte index	Description	Type	Unit
0	Frame character '#'	uint8	-
1	X 0	float32	V
2			
3			
4			
5	V_s 0	float32	()
6			
7			
8			
9	T_w 0	float32	deg K
10			
11			
12			
13	T_a 0	float32	deg K
14			
15			
16			
17	X 1	float32	V
18			
19			
20			
21	V_s 1	float32	()
22			
23			
24			
25	T_w 1	float32	deg K
26			
27			

28			
29	T_a 1	float32	deg K
30			
31			
32			
33	X 2	float32	V
34			
35			
36			
37	V_s 2	float32	()
38			
39			
40			
41	T_w 2	float32	deg K
42			
43			
44			
45	T_a 2	float32	deg K
46			
47			
48			
49	X 3	float32	V
50			
51			
52			
53	V_s 3	float32	()
54			
55			
56			
57	T_w 3	float32	deg K
58			
59			
60			
61	T_a 3	float32	deg K
62			
63			
64			
65	X 4	float32	V
66			
67			
68			
69	V_s 4	float32	()
70			
71			
72			
73	T_w 4	float32	deg K
74			
75			
76			
77	T_a 4	float32	deg K
78			
79			
80			
81	X 5	float32	V
82			

83			
84			
85	V_s 5	float32	()
86			
87			
88			
89	T_w 5	float32	deg K
90			
91			
92			
93	T_a 5	float32	deg K
94			
95			
96			
97	X 6	float32	V
98			
99			
100			
101	V_s 6	float32	()
102			
103			
104			
105	T_w 6	float32	deg K
106			
107			
108			
109	T_a 6	float32	deg K
110			
111			
112			
113	X 7	float32	V
114			
115			
116			
117	V_s 7	float32	()
118			
119			
120			
121	T_w 7	float32	deg K
122			
123			
124			
125	T_a 7	float32	deg K
126			
127			
128			
129	CRC16-CCITT	uint16	-
130			

*CRC16-CCITT 0x1021 polynomial. Initial value = 0xFFFF

IMPORTANT NOTE: Data are transmitted using the little-endian convention, so that the first byte transmitted for each quantity is the least significant.

Checksum & data corruption warning
A CRC-16 checksum word (uint16) is included at the end of each data packet to provide a warning of data loss or corruption in transmission. Example C++ code and DLL files to compute the CRC-16 checksum are available upon request. If the computed and transmitted checksums do not match, the entire data packet should be discarded.

Note that additional details about the probe communications, including a summary of CRC-16-CCITT implementation, are appended to the end of this document.

4 CARE AND HANDLING

WARNING: Do not allow any liquids to come into contact with the sensor package or data acquisition unit, or the system may be permanently damaged.

WARNING: This system is intended for use with SSL proprietary CMOS sensors, and cannot be used with platinum-tungsten hot-wire probes.

- ESD precautions should be taken before handling.
- Always wear powder-free Latex or Nitrile gloves when handling bare boards.
- Unless specifically supplied for use in harsh environments, sensors must be kept free of dust, dirt and liquid. This will alter the system performance.
- Protect the sensor package from moisture and dust, and store in an ESD-safe sleeve when not in service.
- Connect cables to the probe with care, as the socket mountings are fragile.
- Ensure that appropriate strain relief is used: cable strain may cause erroneous sensor readings.
- Do not use the standard sensors in wet or condensing conditions. Store in dry environment, or with desiccant pouch. Coated sensors are available for use in wet, conductive or other harsh environments.

5 INSTALLATION

WARNING: The sensing elements are fragile, and should be handled with care.

Sensor-data acquisition unit interface: Each sensing unit is connected to the data acquisition unit with a 4-way Molex Pico-Lock cable assembly via the ports shown in Figure 1.

Take care when connecting Pico-Lock cables, do not apply any considerable force or bending moment to socket. Magnifying glass may be required to attain correct orientation. With the board facing upwards (components top side), the correct plug orientation is obtained by having the (very) small locking tab near the end of the plug facing upwards. If the connector does not go in, DO NOT APPLY FORCE, but try again with orientation flipped.

Power and data connections: The data acquisition unit draws power from both the DC supply and communications line. There are two user-accessible ports on the unit- a micro-USB port and a serial comms port.

Micro-USB port: This allows the user to access the sensing and diagnostic functions of the probe system using a PC (with the appropriate drivers and software installed).

Comms port: This is the TTL serial communications connection. There are four pins: +5V (1), GND (2), Tx (3) and Rx (4), where pin 1 is on the left when the probe is oriented such that the serial port is below the micro-USB port.

IMPORTANT: The Tx line described is the transmit bus for the sensor package. This should be connected to the Rx of the unit receiving the data.

Note excess voltage (greater than 5V) in the TTL port will cause damage to the system.

6 SOFTWARE AND DRIVERS

Software is included to interface with a PC via the USB port, for data logging, diagnostics and visualization.

Drivers

There are two external drivers which must be downloaded and installed on the computer in order for the PC to be able to interface with the probe system, in addition to the specific system driver for your probe.

- National Instruments LabView Run Time Engine (LVRTE)
- National Instruments VISA Run-Time Engine (NIVISA)

These drivers are freely available for download from National Instruments. Compatible versions of both of these drivers have been bundled with your software package.

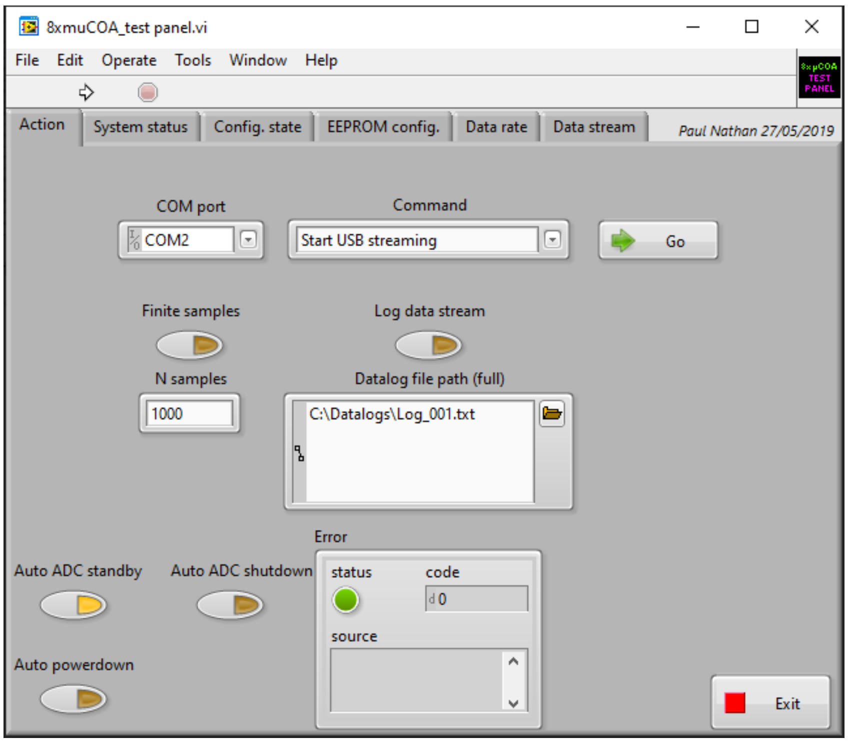
Executable

An executable software package is provided with your probe system, which facilitates direct communication between your PC and your probe using the probe's micro-USB connector. After launching the software, you should see the window reproduced below (Figure 4**Error! Reference source not found.**). Figure 3 shows an example of the software running with USB streaming enabled.

Starting procedure

1) Connect the computer to the system's micro-USB socket.
2) The system will perform a power-on self-test, lasting a couple of seconds. If all tests pass, the on-board green LED will illuminate.
3) Load the program. It will start in the [ACTION] tab.
4) Using the [COM PORT] drop down menu, select the appropriate COM port.
5) Select the desired data rate, chart timespan and datalog duration, and enter a data file path if logging is enabled.
6) Press the white arrow near the top left corner of the window to run the application.

7) Ensure that there are no errors in the [ERROR] dialogue box. If an error has occurred at this stage, it is most likely an invalid COM port selection.



Paul Nathan 27/05/2019

COM port: COM2

Command: Start USB streaming

Finite samples

Log data stream

N samples: 1000

Datalog file path (full): C:\Datalogs\Log_001.txt

Auto ADC standby

Auto ADC shutdown

Auto powerdown

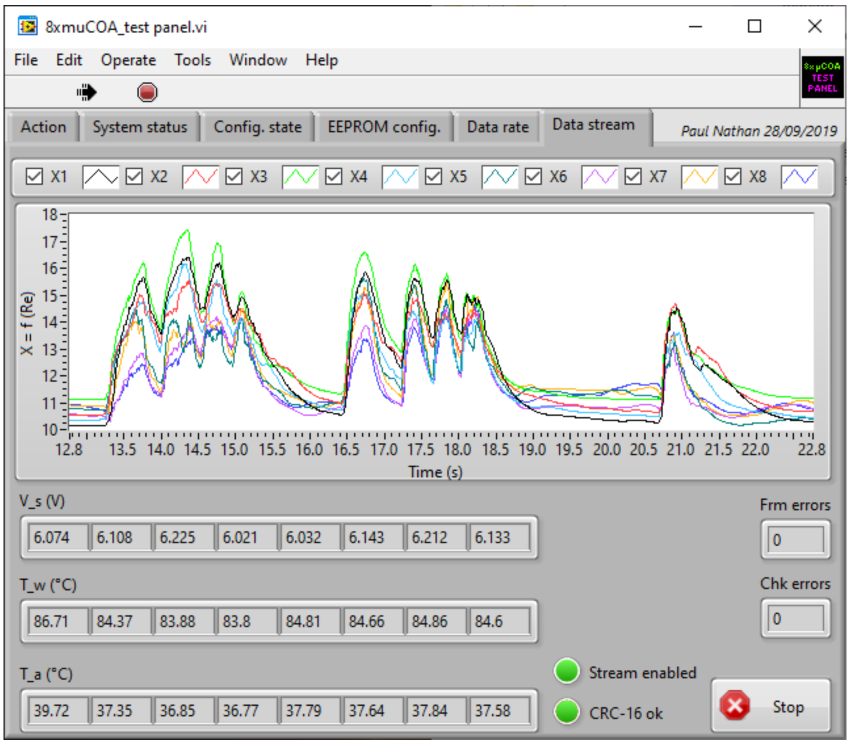
Error

status

code: 0

source

Figure 4: System testing software screenshot



Paul Nathan 28/09/2019

V_s (V): 6.074 | 6.108 | 6.225 | 6.021 | 6.032 | 6.143 | 6.212 | 6.133

T_w (°C): 86.71 | 84.37 | 83.88 | 83.8 | 84.81 | 84.66 | 84.86 | 84.6

T_a (°C): 39.72 | 37.35 | 36.85 | 36.77 | 37.79 | 37.64 | 37.84 | 37.58

Frm errors: 0

Chk errors: 0

Stream enabled

CRC-16 ok

Figure 3: USB stream screen shot

7 TECHNICAL SUPPORT

Full technical support is available for this product and its associated software.

If you experience any difficulty in installation or use, or if you need additional support in the operation of the system, please contact your Surrey Sensors Ltd. account manager or technical representative.

Appendix: Additional communications details

Status bytes map

Byte num.	Bit	Description	Info
0	0	Channel 0 voltage in range	1 yes, 0 no
	1	Channel 1 voltage in range	1 yes, 0 no
	2	Channel 2 voltage in range	1 yes, 0 no
	3	Channel 3 voltage in range	1 yes, 0 no
	4	Channel 4 voltage in range	1 yes, 0 no
	5	Channel 5 voltage in range	1 yes, 0 no
	6	Channel 6 voltage in range	1 yes, 0 no
	7	Channel 7 voltage in range	1 yes, 0 no
1	0	Channel 0 T_w in range	1 yes, 0 no
	1	Channel 1 T_w in range	1 yes, 0 no
	2	Channel 2 T_w in range	1 yes, 0 no
	3	Channel 3 T_w in range	1 yes, 0 no
	4	Channel 4 T_w in range	1 yes, 0 no
	5	Channel 5 T_w in range	1 yes, 0 no
	6	Channel 6 T_w in range	1 yes, 0 no
	7	Channel 7 T_w in range	1 yes, 0 no
2	0	Channel 0 T_a in range	1 yes, 0 no
	1	Channel 1 T_a in range	1 yes, 0 no
	2	Channel 2 T_a in range	1 yes, 0 no
	3	Channel 3 T_a in range	1 yes, 0 no
	4	Channel 4 T_a in range	1 yes, 0 no
	5	Channel 5 T_a in range	1 yes, 0 no
	6	Channel 6 T_a in range	1 yes, 0 no
	7	Channel 7 T_a in range	1 yes, 0 no

EEPROM Map

Byte index	Description	Type	Unit	Notes
0	Hot resistance (hardware)	float32	ohms	
1				
2				
3				
4	Default sampling rate	float32	Hz	Maximum 3125 Hz
5				
6				
7				
8	Fluid medium	uint8	-	0 = air; 1 = water (pure), 2 = seawater (3.5% salinity)
9	Vscale 0	float32	()	Nominally 1.3
10				
11				
12				
13	Vscale 1	float32	()	Nominally 1.3
14				
15				
16				
17	Vscale 2	float32	()	Nominally 1.3

18				
19				
20				
21	Vscale 3	float32	()	Nominally 1.3
22				
23				
24				
25	Vscale 4	float32	()	Nominally 1.3
26				
27				
28				
29	Vscale 5	float32	()	Nominally 1.3
30				
31				
32				
33	Vscale 6	float32	()	Nominally 1.3
34				
35				
36				
37	Vscale 7	float32	()	Nominally 1.3
38				
39				
40				
41	Vscale 8	float32	()	Nominally 1.3
42				
43				
44				
45	Vscale 9	float32	()	Nominally 1.3
46				
47				
48				
49	Vscale 10	float32	()	Nominally 1.3
50				
51				
52				
53	Vscale 11	float32	()	Nominally 1.3
54				
55				
56				
57	Vscale 12	float32	()	Nominally 1.3
58				
59				
60				
61	Vscale 13	float32	()	Nominally 1.3
62				
63				
64				
65	Vscale 14	float32	()	Nominally 1.3
66				
67				
68				
69	Vscale 15	float32	()	Nominally 1.3
70				
71				
72				
73	CRC16-CCITT	uint16	-	CRC16-CCITT 0x1021 polynomial. Initial value = 0xFFFF
74				

USB Command list

ASCII Cmd. Str. After '@'	Description	Return	Additional information	Comments
^	One-shot data	130x uint8	{'#', [X[4], V_s[4], T_w[4], T_a[4]]_0...7, CheckSum[1]}	
$	Start USB data streaming	130x uint8	{'#', [X[4], V_s[4], T_w[4], T_a[4]]_0...7, CheckSum[1]}	Repeated every [Datarate] us until stopped by '|' cmd.
|	Stop USB data streaming	-		
S	Enter standby mode	-		
s	Exit standby mode	-		
Z	Enter shutdown mode	-		
z	Exit shutdown mode	-		
P	Enable system power	-		
p	Disable system power	-		
D	Set data rate (Hz)	-	Send byte array as {'D', x[4]}, where x[0…3] is the float32 sampling rate in Hz	
E	Set EEPROM values	-	Send byte array as {'E', R_set_h[4], Default_datarate[4], fluid_medium[1], CheckSum[1]}	
e	Get EEPROM values	10x uint8	{R_set_h[4], Default_datarate[4], Fluid_medium[1], CheckSum[1]}	Fluid medium: 0 = air, 1 = water
Q	Get configuration state	8x uint8	{Data_rate[4], Data_streaming[1], ADC_mode[1], System_power[1], Error_state[1]}	ADC_mode: 0 = shutdown, 1 = standby, 2 = active
C	Clear error LED	-		
R	System reset	-		
?	Get system status bytes	3x uint8	{V_s[0…7], T_w[0…7], T_a[0…7]}	0 = fail, 1 = pass

Interpretation of the Output Variable "X"

The datastream contains the following output variables:

X	()	Non-dimensional temperature corrected output
V_s	(V)	Sensor drive voltage
T_w	(°C)	Sensor element temperature
T_a	(°C)	Ambient temperature

The sensor drive voltage is output should the end-user wish to do their own temperature corrections. Saturation occurs around 13.5V. The end-user should check that this condition is not present at any time.

At very low flow speeds the ambient temperature reading will be affected by the proximity of the hot sensor element, as well as heat transfer from the elements on the circuit board. This results in an overreading of the ambient temperature.

X is the variable that should be used when calibrating against flow speed. It is defined simply as follows

$$X = f\,(\mathrm{Re})$$

This can be written alternatively as

$$\mathrm{Re} = g\,(X)$$

$$\frac{\rho D d}{\mu} = g\,(X) \tag{1}$$

where ρ is the density (kg/m^3), μ is the dynamic viscosity ($Pa \cdot s$) and d is the length scale of the sensor element, which is taken to be 3.0 x 10^{-4} m. The fluid properties are evaluated at the *film temperature* T_f which is defined as

$$T_f = \frac{1}{2}\,(T_w + T_a) \tag{2}$$

The function *g(X)* can be chosen as appropriate to give the best fit to the obtained calibration data points. Typically a 5th order polynomial is sufficient.

<u>Calibration data fitting procedure:</u>

1. Plot a graph of $\frac{\rho D d}{\mu}$ against X , with X as the independent variable

2. Obtain *g(X)* by a least-squares method or non-linear regression as appropriate

Now the flow speed can be measured by rearranging equation (1) as follows

$$U = \frac{\mu}{\rho d} \cdot g\,(X\,) \tag{3}$$

where the density and viscosity are evaluated at the film temperature at the time of measurement.

Supposing a 5th order polynomial, then equation (3) can be expressed in Horner Form for efficient evaluation as

$$U = \frac{\mu}{\rho d}\left(a_0 + X\left(a_1 + X\left(a_2 + X\left(a_3 + X(a_4 + Xa_5)\right)\right)\right)\right) \tag{4}$$

Note that the length scale *d* may be omitted and absorbed into the polynomial coefficients. However keeping it separate improves the numerical conditioning of the coefficients – particularly important on limited precision embedded systems.

Empirical functions for the density and viscosity of (dry) air can be formed using data in, for example, the CRC Handbook of Chemistry and Physics. Alternatively, the ideal gas law can be used to obtain the density and Sutherland's formula can be used for the viscosity:

$$\rho(P,T) = \frac{P}{RT}$$

$$\mu(T) = \mu_0 \left(\frac{T}{T_0}\right)^{3/2} \frac{T_0 + S}{T + S}$$

with the constants:

R = 287.058 *J/kgK*

μ_0 = 1.716 x 10^{-5} *Pa · s*

T_0 = 273.15 *K*

S = 110.4 *K*

The ambient relative humidity will have a small effect on the air density. For example at 25ºC and 1 bar, 99.99% relative humidity causes a 1.2% reduction in air density relative to dry air.

Summary of CRC-16-CCITT Implementation in C++

Global Variables and Constants

```
uint16_t CRC16_LUT[256];
const uint16_t poly = 0x1021;
const uint16_t crc_init = 0xFFFF;
```

CRC-16 Lookup Table (LUT) Generation

The following function is called once at the start. The 1D array of length 256 “CRC16_LUT” is then stored in memory for all time and used whenever a CRC is computed.

```
void Generate_CRC16_LUT()
{
        for (uint16_t i = 0; i < 256; i++)
        {
                uint16_t Byte = i << 8;

                for (uint8_t Bit = 0; Bit < 8; Bit++)
                {
                        if ((Byte & 0x8000) != 0)
                        {
                                Byte <<= 1;
                                Byte ^= poly;
                        }
                        else
                        {
                                Byte <<= 1;
                        }
                }

                CRC16_LUT[i] = Byte;
        }
}
```

Alternatively, the LUT can be hard-coded as a constant:

```
// CRC-16 lookup table for CCITT polynomial 0x1021

static const uint16_t CRC16_LUT[256] =
{
  0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
  0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
  0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
  0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
  0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
  0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
  0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
  0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
  0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
  0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
  0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
  0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
  0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
  0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
  0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
  0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
  0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
  0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
  0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
  0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
  0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
  0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
  0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
  0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
  0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
  0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
  0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
  0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
  0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
  0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
  0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
  0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
};
```

CRC-16 Computation

The following function is called whenever a CRC-16 is required from an array of data.

```
uint16_t Calc_CRC16(uint8_t *Data, uint16_t DataLen, uint16_t crc)
{
       for (uint16_t i = 0; i < DataLen; i++)
       {
              uint8_t index = Data[i] ^ (crc >> 8);

              crc = CRC16_LUT[index] ^ (crc << 8);
       }

       return crc;
}
```

CRC-16 Function Call Example

The data for which the CRC is to be computed is first of all typecast into an array of unsigned char (uint8_t) "DataBytes". This can be done using the `memcpy` function. When generating a CRC value for an array of data the length value "Len" passed to the function is that of the number of bytes in the entire array. However, when checking a CRC value appended to an array of data, the length value passed to the function is two less than that of the entire array so as to exclude the appended CRC word. The CRC value passed to the function is that of the initialiser constant "crc_init", which, for the CCITT specification, is hexadecimal `0xFFFF`.

```
uint16_t CRC_computed = Calc_CRC16(&DataBytes, Len, crc_init);
```

Checksum Test

A checksum test is passed if the computed and transmitted checksum values are equal. With the CRC appended at the end of the transmitted data array the test is carried out as follows

```
uint16_t CRC_appended;

memcpy(&CRC_appended, &DataBytes[Len - 2], 2);

bool CRC_pass = (CRC_appended == CRC_computed);
```

Code implementation can be validated by cross-checking results with a reputable online CRC calculator such as https://crccalc.com/

CRC-16-CCITT Algorithm Parameters:

Polynomial divisor:	`0x1021`	$(x^{16} + x^{12} + x^{5} + 1)$
CRC initialiser:	`0xFFFF`	
Input reflection:	`False`	
Output reflection:	`False`	
Output XOR:	`0x0000`	