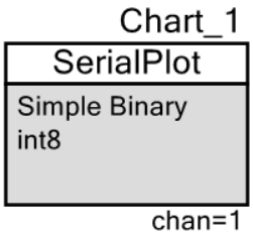
SerialPlot: interface to real-time data charts
0.0

Features

- Implements interface between PSoC and SerialPlot software
- Data types: int8, int16, int32, uint8, uint16, uint32, float
- Doesn't use hardware resources
- Up to 8 channels



General description

The SerialPlot[*] component implements interface to the real-time charting software SerialPlot [1]. Using this component, PSoC data can be easily visualized on personal computer. Component doesn't consume hardware resources, performing all operations by CPU, which is useful for systems with little resources, such as PoC4. Multiple instances of the component can run simultaneously in the project.

When to use SerialPlot component

Component was developed for monitoring temperature profile from several sensors. It can be useful whenever digitized signals need to be visualized or saved on external computer, such as: voltage, current or temperature monitoring, PID parameters tune-up, etc. Component is useful for a system with limited hardware resources, such as PSoC4. Component was tested using CY8KIT-059 PSoC5LP Prototyping Kit and CY8KIT-042 PSoC4 Pioneer Kit. Demo projects are provided.

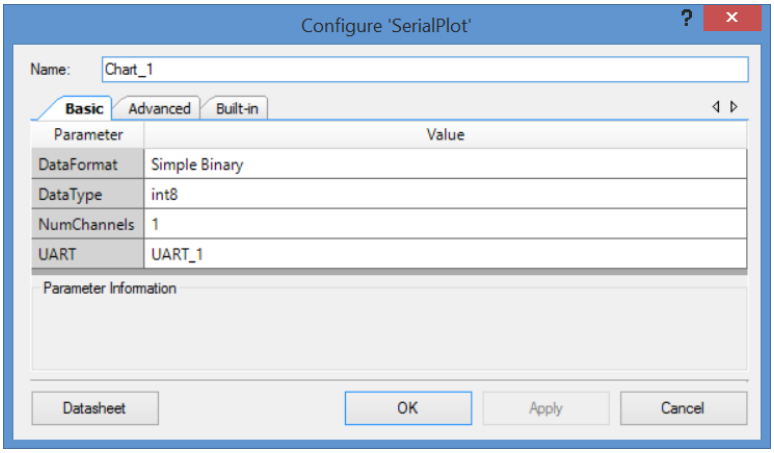
[*] Hereafter referred to as "Chart"

Input-output connections

Component does not have any input/output connections. It is, in essence, a software library, performing all operations by API. Unlike library, multiple instances of the component can run simultaneously in the project.

Parameters and Settings

Basic dialog provides following parameters(*):



DataFormat [Simple Binary / ASCII / Custom Frame]

Sets communication format. Valid options are: Simple Binary, and ASCII, Custom Frame. Default setting is Simple Binary. Value can't be changed during the run-time.

DataType [int8 / int16 / int32 / uint8 / uint16 / uint32 / float]

Sets data type. Valid options are int8, int16, int32, uint8, uint16, uint32, and float. Default value int8. The value can't be changed during the run-time.

NumChannels (uint8)

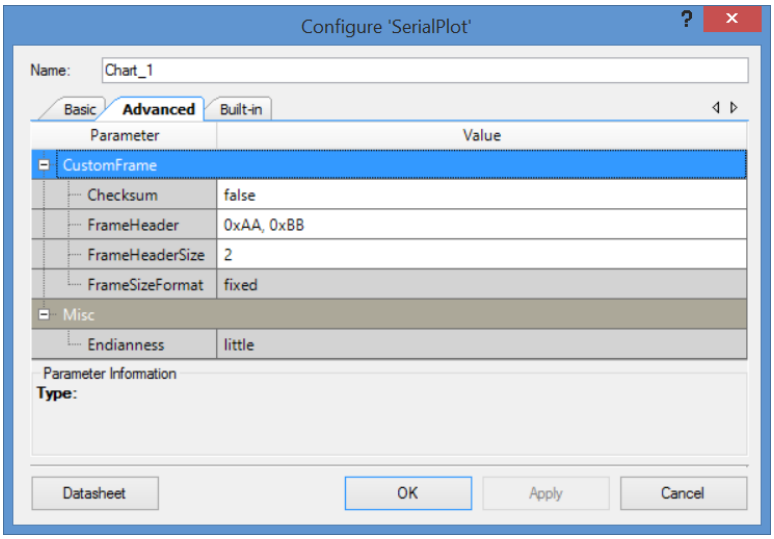
Sets number of output channels. Each channel appears as a separate line on the chart. Valid options are [1 to 8]. Default value is 1. The value can't be changed during the run-time.

UART (string)

Sets name of the complementary UART component, which is used for communication with SerialPlot software. Component doesn't have UART built-in; it must be added to the project. For full functionality, the UART selected must provide access to procedures PutChar(), PutArray() and PutString(). Default name is UART_1. The value can't be changed during the run-time.

* Component was intentionally compiled using Creator 4.0 for compatibility with older versions.

Advanced dialog provides following parameters:



CheckSum (bool)

Enables checksum calculation. This option has effect only in the Custom Frame mode. Enabling checksum increases transmission reliability but adds some overhead. Default value is false. The value can't be changed during the run-time. See **Implementation** section for details.

FrameHeader (string)

Sets frame header[*] preamble. Frame header is a unique set of bytes inserted at the start of each data frame, which is used for synchronization by the SerialPlot. This option has effect only in the Custom Frame mode. The value should be typed as string of 8-bit Hex characters, separated by the comma. The length of the header can be in the range between 2 to 8 bytes. Longer header size improves transmission reliability, but reduces communication speed. Default value is "0xAA, 0xBB" (2 bytes long). The value can't be changed during the run-time.

FrameHeaderSize (uint8)

Sets frame header size. This option has effect only in the Custom Frame mode. The value must match the FrameHeader string. Default value is 2. The value can't be changed during the run-time.

FrameSizeFormat [fixed]

This version of Component supports only fixed frame format. This parameter can't be changed.

[*] In the SerialPlot software it is also called "Frame Start"

Endianness [little]

PSoC devices are little endian. This parameter can’t be changed.

Application Programming Interface

Function	Description
`Chart_Plot()`	Sends single data packet

void Chart_Plot(data_t V1, data_t V2, … , data_t Vn)

Description: Sends one set (column) of data samples to the COM port.

Parameters: input data. The **data_t** type represents one of the selected data types: int8, int16, int32, uint8, uint16, uint32 or float32. The number of the arguments in the list equals the number of channels. Upon compiling the project, component generates overload version of the Plot() procedure based on the Dialog settings. See **Implementation** section for details.

Return Value: none

Functional Description

The SerialPlot is open source real-time plotting software, which graphically displays data received by computer through a serial port. It is available for Linux and Windows platforms, and supports several data formats with multichannel operation. The program can receive data as: (i) simple binary data stream; (ii) ASCII data in CSV format; (iii) user-defined custom frame format. The SerialPlot software is ideal tool for displaying data produced by microcontrollers, such as various sensors outputs, control and debugging information (Figure 1).

The Chart component implements easy interface solution between PSoC microcontrollers and the SerialPlot software using UART communication. The component doesn't include any UART blocks by itself, requiring complimentary external UART for operation. The Chart component merely formats the data for the UART communication buffer, while UART handles actual transmission of the data. Such separation allows for greater flexibility in the UART selection.

The Chart component supports many, but not all of the features available on the SerialPlot software. It is designed to simplify data output from PSoC microprocessors to the SerialPlot software running on personal computers. Upon compiling the project it will automatically generate proper overload version of the Plot() procedure, based on the options selected in the Dialog.

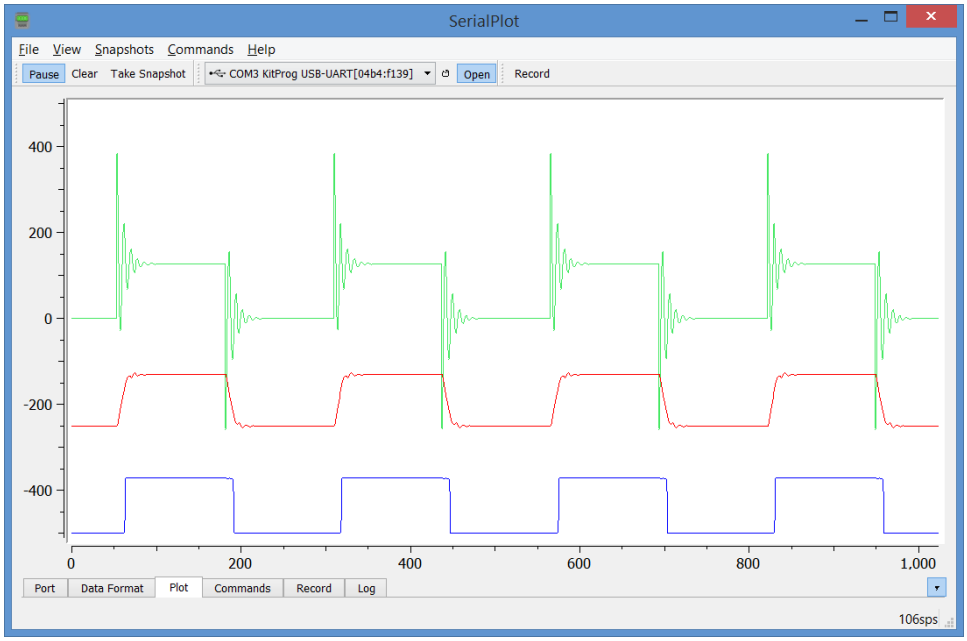


Figure 1. Example of the SerialPlot output of data received through a serial port.

Implementation

The Simple Binary format

In the Simple Binary format, data is transmitted as a stream of data packets, one at a time. Data packet structure for single- and dual-channels transmission is shown of Figure 2.

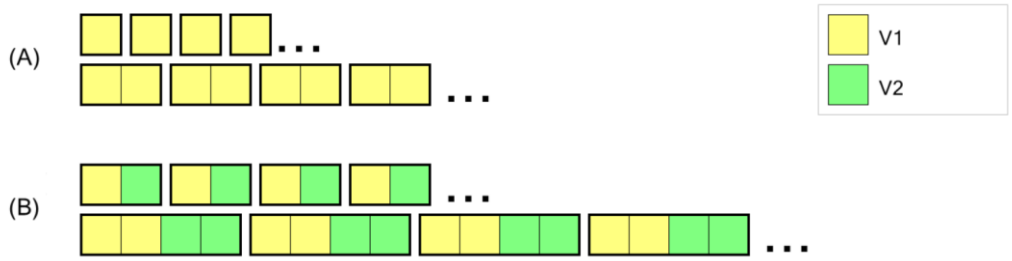


Figure 2. Structure of data packets in the Simple Binary format, each square represents a byte: (A)- single channel of int8 / int16 data type; (B)- 2 channels of int8 / int16 data type.

There is no option to synchronize the data stream in this mode – the loss of a single byte catastrophically corrupts the rest of the data received. The only exception presents 1-byte single channel transmission, where each byte corresponds to the individual data point. For that reason the Single Binary mode is recommended only when data packet fits one byte - a single channel of int8 or uint8 data.

Upon compiling the project, the component automatically generates overload version of the Plot() procedure, based on the Dialog settings (see **Appendix 1**, Simple Binary section).

The ASCII format

In the ASCII format, received data is formatted as a string of human-readable values, separated by the comma, with each string is terminated by the standard escape sequence: [CR][LF]. For example, when configured for the 2-channels of float data type, the output string looks like:

-1.23456E+01, 2.34567E+02[CR][LF]

Due to hard-separation of the data strings by the escape sequence, this method of data transmission is relatively immune: a loss of a data byte doesn't corrupt the rest of data. This format is slow and CPU-hungry (see **Performance** section) due to conversion of the data into human-readable form and not recommended for plotting. It may be useful, however, for debugging purposes using character Terminal. See **Appendix 1**, ASCII section for details.

The Custom Frame format

In the Custom Frame format[*], the data is packed into individual frames, preceded with a frame header[†], a column of data, and optional checksum byte at the end of the frame. Each frame is transmitted separately. It offers performance comparable to the Single Binary with robustness of the ASCII format.

The frame header allows re-synchronizing data stream if some bytes are lost or corrupted. It consists of several unique user-selectable characters (up to 8 bytes long), for example: "0xAA, 0xBB, 0xCC, 0xDD", which would rarely occur naturally. Longer header strengthens sync lock, but adds overhead. The default header "0xAA, 0xBB" is 2-byte long, which is sufficient for non-demanding applications. For example, when the number of channels is 2 and selected data type is int16 (2-byte), the frame structure is:

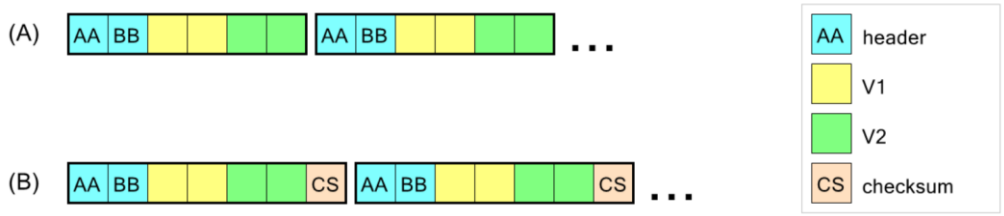


Figure 3. Example of the data packet in the Custom Frame format, where each square represents a byte. Header size is 2 bytes, number of channels is 2, data type is int16. (A)- checksum disabled, (B)- checksum enabled.

Optional checksum allows protection against data being lost or corrupted. If checksum doesn't match, the entire frame is discarded by the SerialPlot software. The checksum byte is obtained by simply summing all data bytes in the transmitted array. See **Appendix 1**, Custom Frame section for details.

Features not implemented

- Multiple data packets per frame in Custom Frame format.
- Variable frame length in Custom Frame format
- Bidirectional communication (Rx + Tx)
- USB-UART communication

[*] This simplified implementation is limited to only Fixed size format with a single data column per frame.
[†] Also called as "Frame Start".

Performance

Component was tested using CY8KIT-059 PSoC5LP Prototyping Kit and CY8KIT-042 PSoC4 Pioneer Kit. The component performs all operation entirely by the CPU. Results for PSoC5LP are presented below(*). Results for PSoC4 are typically ~20% slower.

The API execution time in binary modes is clearly dominated by the UART performance, while in ASCII mode by the string formatting procedure. The fastest time is obtained in Single Binary mode with UART Tx buffer size of 4 (hardware FIFO enabled). For larger buffer sizes execution time jumps due to UART switching to the RAM buffer instead of FIFO.

Table 1. Execution time (bus clocks) in Single Binary format, 1 channel.

Tx buffer size	int8	int16	int32	float
4 bytes	17	62	83	83
128 bytes	47	128	196	196

Table 2. Execution time (bus clocks) in Custom Frame format, Header 2 bytes, 2 channels(†).

Tx buffer size	int8	int16	int32	float
4	74	2410	10730	10770
128	185	294	460	455

(†) Enabling checksum adds overhead 50-60 clocks.

Table 3. Execution time (bus clocks) in ASCII format, 2 channels.

Tx buffer size	int8	int16	int32	float
4	12600	17000	31500	45000
128	2560	2600	3100	14000

* Results obtained using UDB UART v2.50 at 115.2 kbd, release mode with compiler optimization set to speed

Resources

Component does not consume hardware resources. It doesn't use interrupts, clocks or UDB. Component does not have built-in DMA capabilities.

Sample Firmware Source Code

Demo project is provided, see **Appendix 2** for details.

Component Changes

Version	Description of changes	Reason for changes/impact
0.0	Version 0.0 is the first beta release of the component	

References

1. Serial PLOT v0.10.0, by Hasan Yavuz Özderya,
   https://hasanyavuz.ozderya.net/?p=244
   https://hackaday.io/project/5334-serialplot-realtime-plotting-software/discussion-88924
   https://bitbucket.org/hyOzd/serialplot
2. PSoC Annotation Library v1.0,
   https://community.cypress.com/thread/48049

Appendix 1

Simple Binary format

When data packet fits a single byte (1-channel of int8 or uint8 data type), the component generates overload version of the Plot() procedure, along with suggested SerialPlot settings:

```
void Chart_1_Plot(int8 V1)
{
    // Format:       Simple Binary
    // Channels:     1
    // Number Type:  int8
    // Endianness:   Little Endian

    UART_1_PutChar(V1);
}
```

Suggested settings should be manually(*) put in the SerialPlot Data Format tab (Figure 4). Recommended size of the complimentary UART Tx buffer in this mode is 4 (hardware FIFO), see **Performance** section for details.

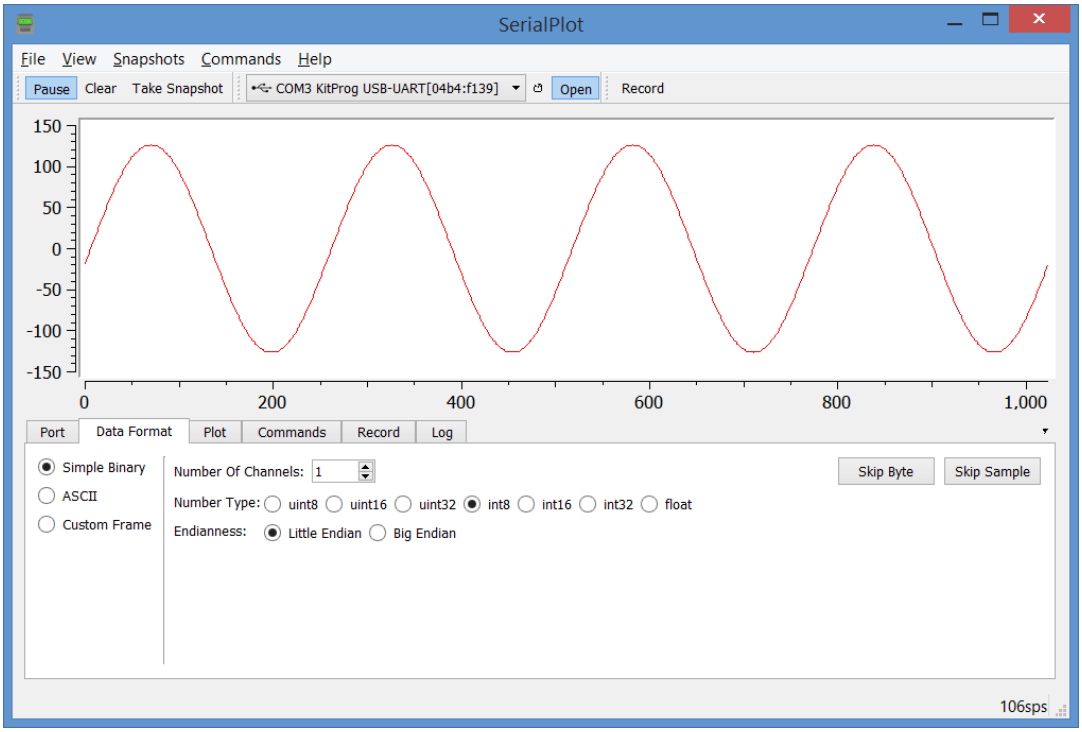


Figure 4. SerialPlot settings for Simple Binary data format, matching component parameters: Number of Channels: 1; Number Type: int8; Endianness: Little Endian.

* Current version of the SerialPlot software (v0.10.0) doesn't support remote configuration.

If data packet doesn't fit a single character (for example 2-channels of int8 data type), component generates overload version of the procedure:

```
void Chart_1_Plot(int8 V1, int8 V2)
{
    // Format:      Simple Binary
    // Channels:    2
    // Number Type: int8
    // Endianness:  Little Endian

    int8 val[2] = {V1, V2};
    UART_1_PutArray((uint8 *) &val, sizeof(val));
}
```

Suggested settings should be manually put in the SerialPlot Data Format tab (Figure 5). The UART Tx buffer size should be set enough to fit all bytes in the data packet. Best performance is achieved when data packet fits UART FIFO buffer (4 bytes). See **Performance** section for details.

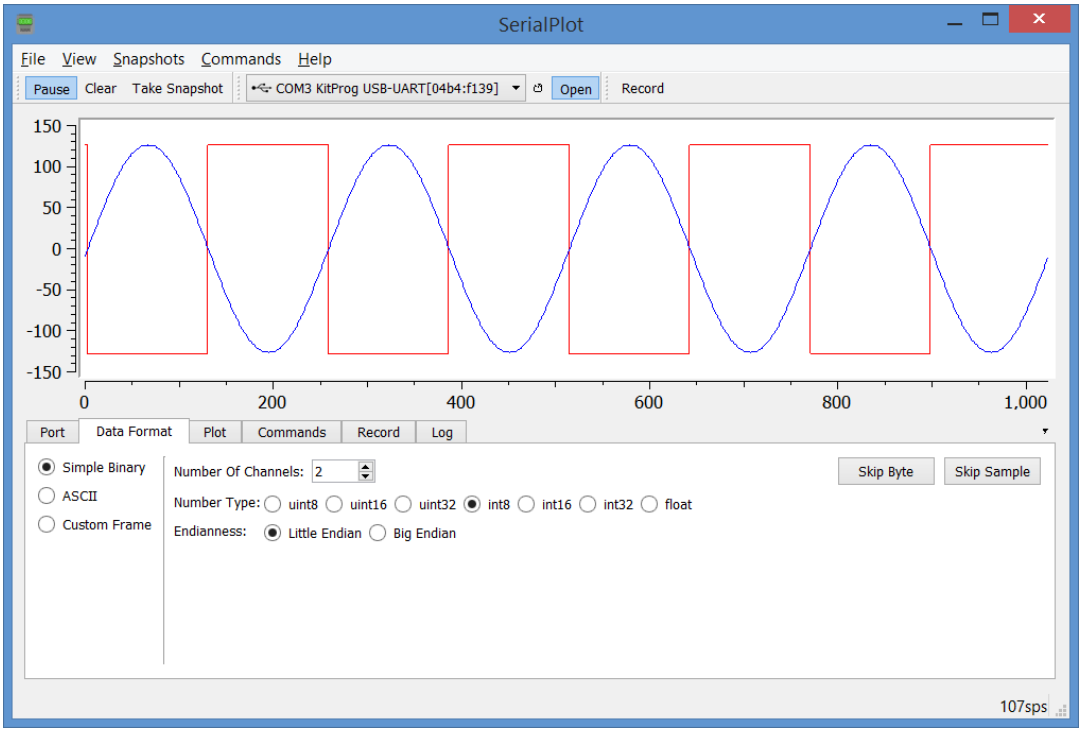


Figure 5. SerialPlot settings for Simple Binary data format, matching component parameters: Number of Channels: 2; Number Type: int8; Endianness: Little Endian.

ASCII format

Upon compiling the project, based on the Dialog settings, the component generates overload version of the Plot() procedure, along with suggested SerialPlot settings:

```
void Chart_1_Plot(float32 V1, float32 V2)
{
    // Format:      ASCII
    // Channels:    2
    // Delimiter:   comma

    char abuff[36];

    sprintf(abuff, "%g,%g\r\n", V1, V2);
    UART_1_PutString(abuff);
}
```

Suggested settings should be manually put in the SerialPlot Data Format tab (Figure 6). When selecting float data format, the newlib-nano Float Formatting must also be enabled in the Project settings. The size of the complimentary UART Tx buffer must be set to no less than the size of the abuff[].

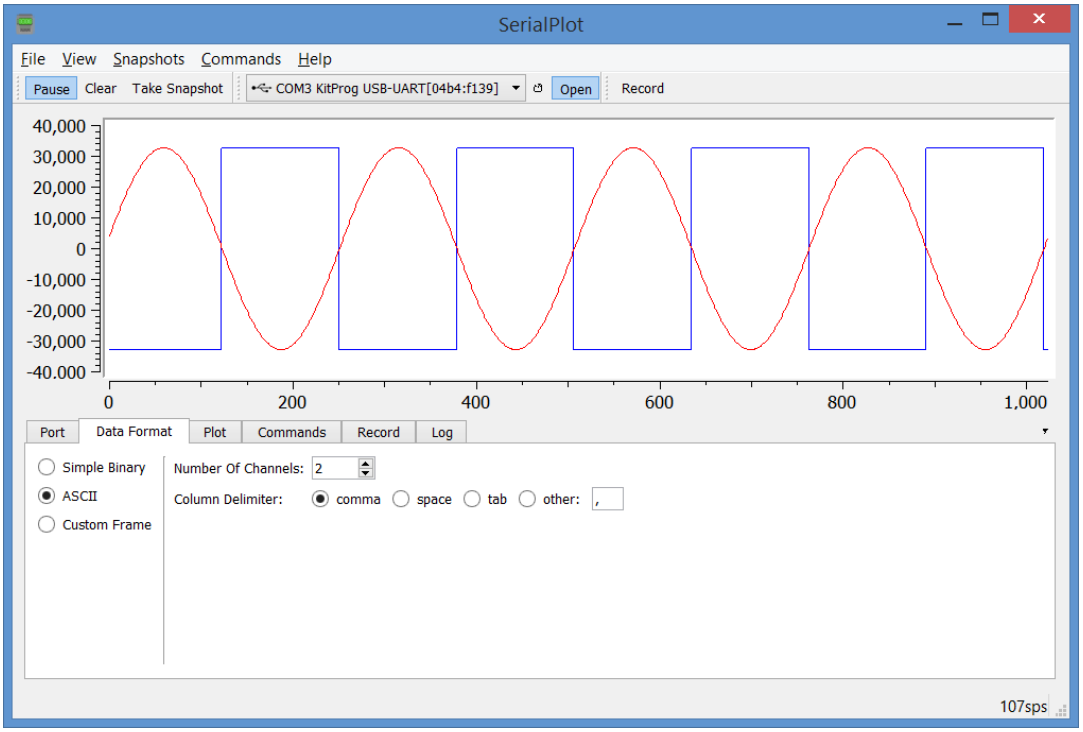


Figure 6. SerialPlot settings for ASCII data format, matching component parameters: Number of Channels: 2; Column Delimiter: comma.

Custom Frame format

The code example of sending 2 channels of the int16-type data in Custom Frame format. Upon compiling the project, the component automatically generates overload version of the procedure, along with corresponding settings for SerialPlot software:

```
void Chart_1_Plot(int16 V1, int16 V2)
{
    // Format:      Custom Frame
    // Frame Start: 0xAA, 0xBB
    // Channels:    2
    // Frame Size:  Fixed, Size=4
    // Number Type: int16
    // Endianness:  Little Endian
    // Checksum:    false

    struct {
        uint8  head[2];
        int16  val[2];
    } __attribute__ ((packed)) Frame  = { {0xAA, 0xBB}, {V1, V2} };

    UART_1_PutArray((uint8 *) &Frame, sizeof(Frame));
}
```

If Checksum validation enabled, the component generates overload version of the procedure, including checksum calculation:

```
void Chart_1_Plot(int16 V1, int16 V2)
{
    // Format:      Custom Frame
    // Frame Start: 0xAA, 0xBB
    // Channels:    2
    // Frame Size:  Fixed, Size=4
    // Number Type: int16
    // Endianness:  Little Endian
    // Checksum:    true

    struct {
        uint8  head[2];
        int16  val[2];
        uint8  CS;
    } __attribute__ ((packed)) Frame  = { {0xAA, 0xBB}, {V1, V2}, 0};

    // calculate checksum
    uint8 * pVal = (uint8 *) &Frame.val[0];
    uint8 * pCS  = (uint8 *) &Frame.CS;
    while (pVal < pCS) { * pCS += * pVal++; }

    UART_1_PutArray((uint8 *) &Frame, sizeof(Frame));
}
```

Settings provided in the comments header should be manually put in the SerialPlot Data Format tab (Figure 7). The size of the complimentary UART Tx buffer in this mode should be set to no less than the frame size.

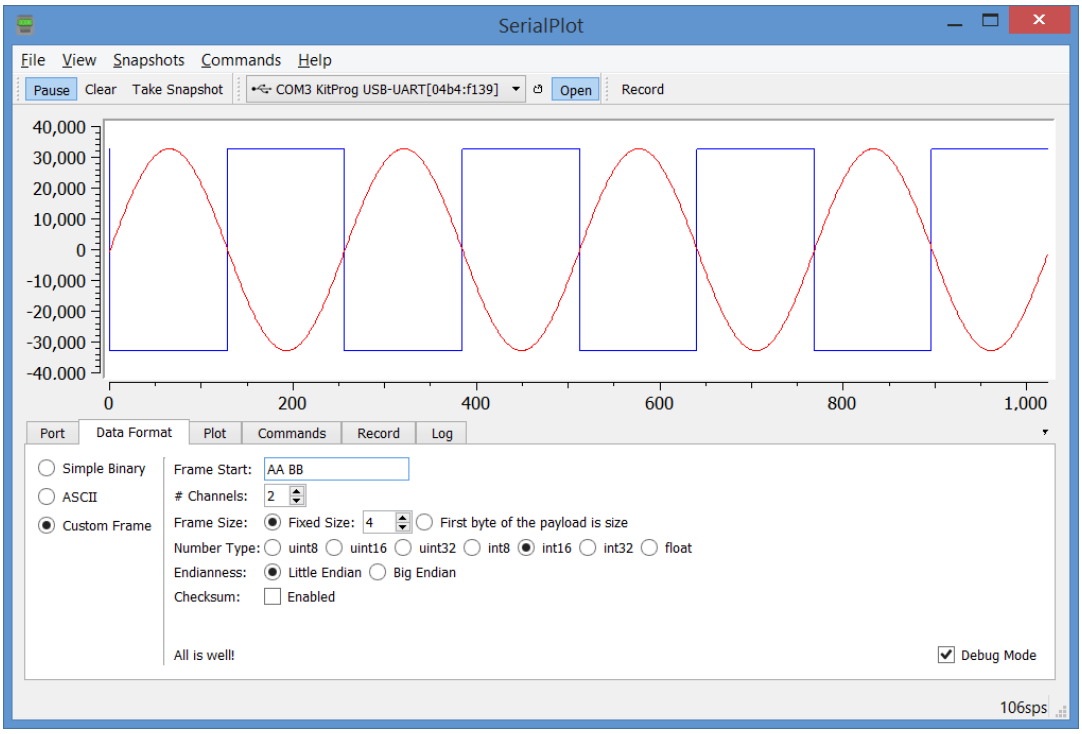


Figure 7. SerialPlot Data Format settings for Custom Frame mode: Frame Start: "0xAA, 0xBB"; #Channels: 2; Frame Size: Fixed, Size=4; Number Type: int16; Checksum Enabled: True.

Appendix 2

Component demo project

The PSoC5 project schematic using SerialPlot component is shown on Figure 8. Test data samples are generated on clock timer and streamed to the host computer using (UDB) UART v2.50 through USB-UART bridge, built into the KitProg.

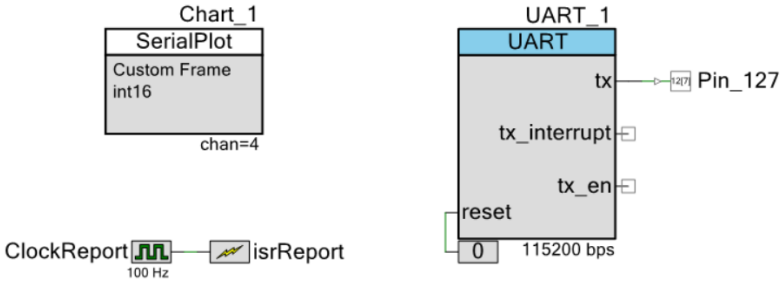


Figure 8. SerialPlot demo project schematic.

The test data are generated on 100 Hz timer clock. The Chart component is configured for Custom Frame format, 4 channels, int16 data type. The SerialPlot software viewport and configuration page are shown on Figure 9. UART Tx buffer size is 128 bytes.

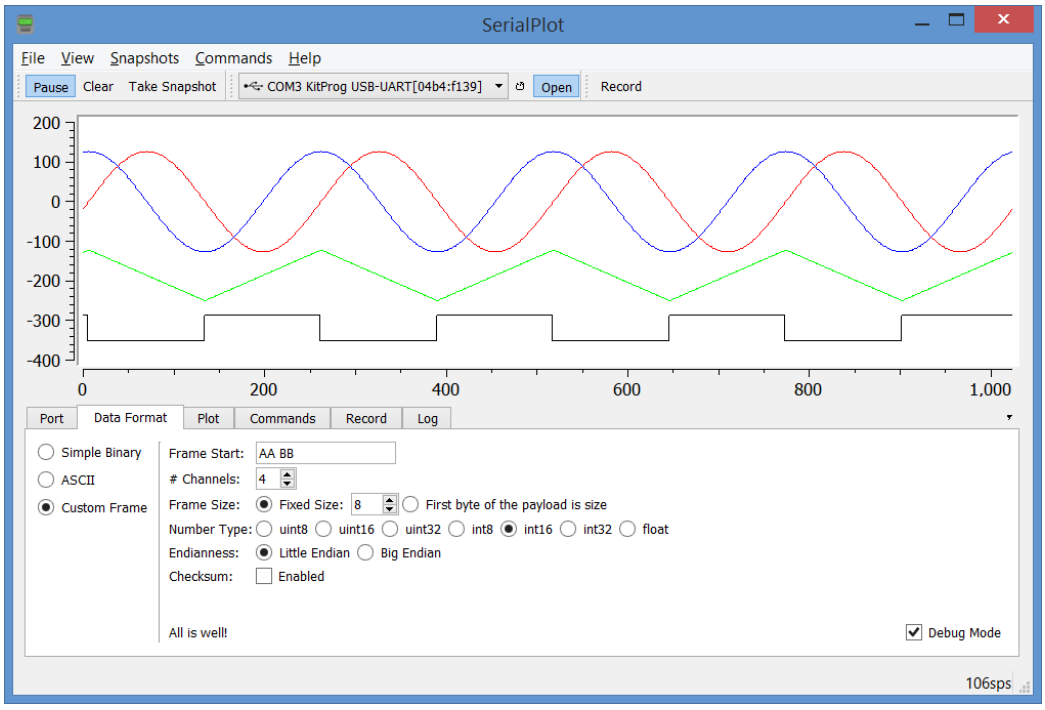


Figure 9. Demo project output and settings. Custom Frame mode: Frame Start: "0xAA, 0xBB"; #Channels: 4; Frame Size: Fixed, Size=4; Number Type: int16; Checksum Enabled: True.

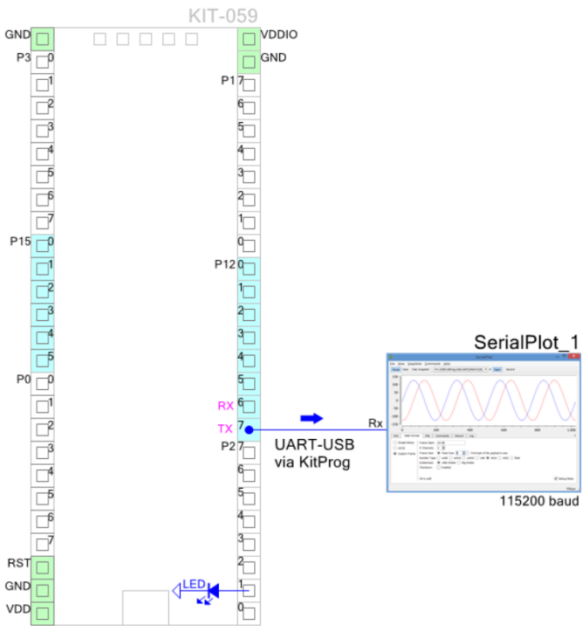


Figure 10. Project annotation for PSoC5 CY8CKIT-059 using PSoC Annotation library[*].

[*] PSoC Annotation Library v1.0, https://community.cypress.com/thread/48049

Appendix 3

Off-chip annotation component

The SerialPlot component is accompanied with off-chip annotation component facilitating schematic drawing and enhancing visibility. It can be used in conjunction with the PSoC Annotation library [2] and KIT-042, KIT-044 and KIT-059 annotation stubs.

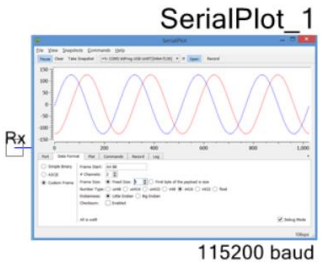


Figure 11. SerialPlot off-chip annotation component.

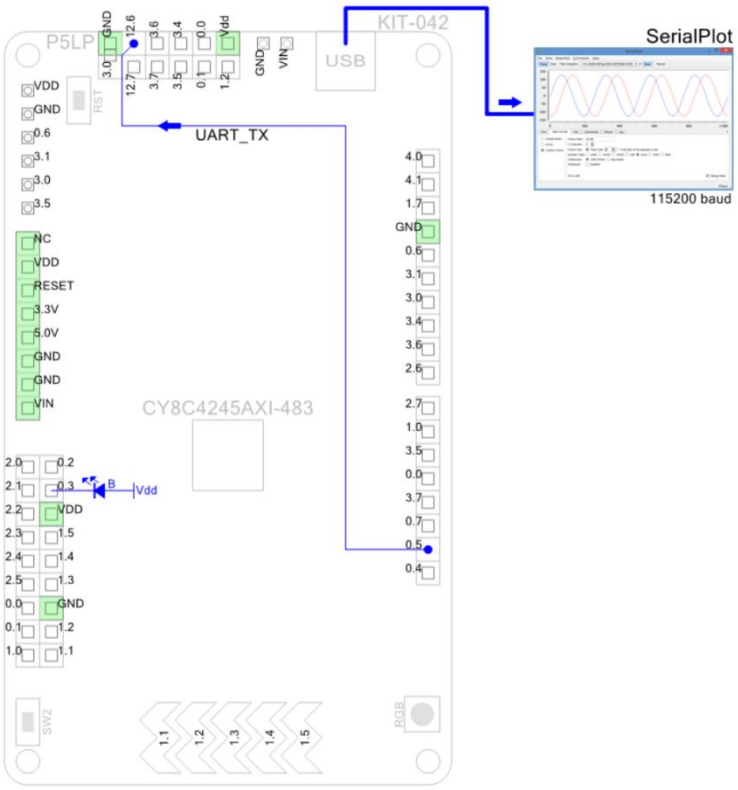


Figure 12. Project annotation for PSoC4 Pioneer Kit (CY8CKIT-042) using KIT-042(*) stub and PSoC Annotation library [2].

* KIT-042: annotation component for CY8CKIT-042 Pioneer Kit, https://community.cypress.com/thread/48741