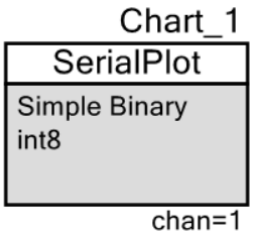
SerialPlot: interface to real-time data charts
1.0

Features

- Implements interface between PSoC and SerialPlot software
- Data types: int8, int16, int32, uint8, uint16, uint32, float
- Doesn't use hardware resources
- Up to 64 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 use any hardware, 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. Component does not have any input/output connections. It is, in essence, a software library, performing all operations by API.

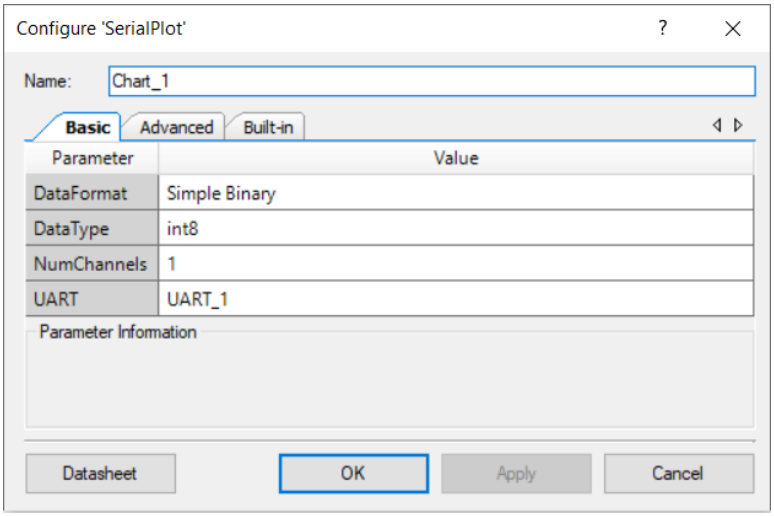
When to use SerialPlot component

Component was developed for monitoring temperature profile from multiple 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, CY8KIT-044 PSoC4200M Pioneer Kit and CY8KIT-042 PSoC4 Pioneer Kit. Demo projects are provided.

[*] Hereafter referred to as "Chart"

Parameters and Settings

Basic dialog provides following parameters[*]:



DataFormat [Simple Binary / ASCII / Custom Frame]

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

NumChannels (uint8)

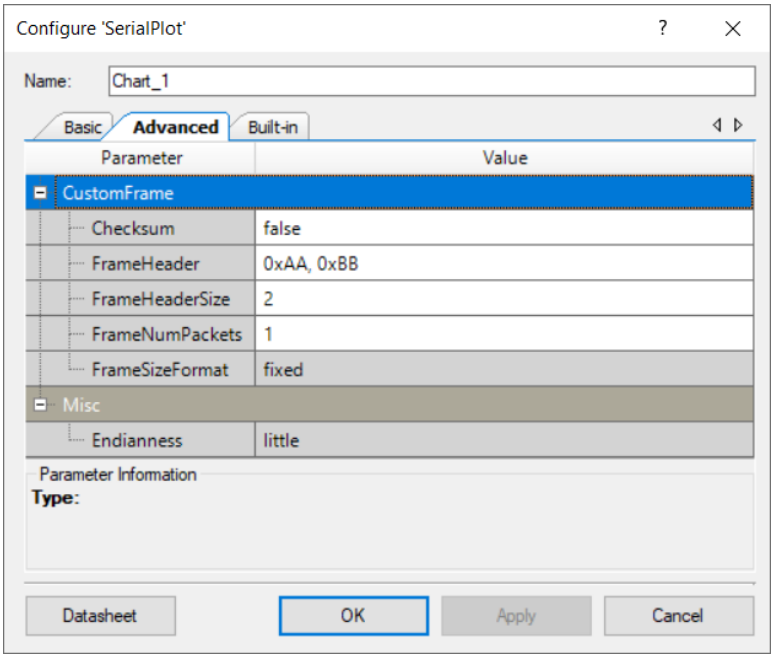
Sets number of output channels. Each channel appears as a separate line on the chart. Valid options are [1 to 64]. The value can't be changed during the run-time. See **API** section for details.

UART (string)

Sets the name of the UART, which transmits data. Component doesn't have any UART built-in. Component works with Creator stock UART and SCB UART components. The setting 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 setting 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 data flow synchronization. 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. The longer header improves transmission reliability, but increases overhead. Default value is "0xAA, 0xBB" (2 bytes long). The setting can't be changed during the run-time.

FrameHeaderSize (uint8)

Sets frame header size in the Custom Frame mode. The value must match the amount of characters in the FrameHeader string. Default value is 2. The value can't be changed during the run-time.

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

FrameNumPackets (uint8)

Sets number of data packets per frame in the Custom Frame mode. Default value is 1. The number of data packets, that can fit into the frame is limited by the maximum frame size of 255 bytes. See **Implementation** section for details. The value can't be changed during the run-time. When the FrameNumPackets is 1, the SerialPlot updates the screen as soon as a new data packet arrives. When the FrameNumPackets is greater than 1, the SerialPlot updates the screen after all data packets in the frame have been received. This helps reducing computer CPU load by lowering the screen update rate.

FrameSizeFormat [fixed]

Current version of the component supports only fixed frame size.

Endianness [little]

The setting is fixed to the little endian. This parameter can't be changed.

Application Programming Interface

Function	Description
`Chart_Plot()`	Sends single data packet
`Chart_PlotArray()`	Sends single data array packet

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

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

Parameters: input data list. The **data_t** type can be one of the following types: int8, int16, int32, uint8, uint16, uint32 or float. The number of the arguments in the list must be equal the number of channels. Upon compiling the project, component generates overload version of the Plot() procedure based on the Dialog settings. The maximum number of channels that can be plotted using this procedure is 16. See **Implementation** section for details.

Return Value: none

void Chart_PlotArray(data_t * channels)

Description: Sends one packet (column) of array data to the COM port.

Parameters: channels - pointer to the data array, where each index represents a channel. The array type must match the selected data type **data_t**, which can be one of the following: int8, int16, int32, uint8, uint16, uint32 or float. The number of values transmitted is defined by the parameter NumChannels, and not by the original array size. The component automatically calculates the amount of bytes to transfer based on the number of channels and selected data type. This amount, however, can't exceed the maximum frame size of 255. The maximum number of channels that can be plotted using this procedure is limited to 64[(*)]. The procedure is not available in the ASCII mode. See **Implementation** section for details.

Return Value: none

[*] The frame size and number of channels are limited by the SerialPlot 0.12.0

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 Windows, Mac and Linux platforms, and supports several data formats and multichannel operation. The application 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. 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 Tx 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 API, based on the options selected in the Dialog.

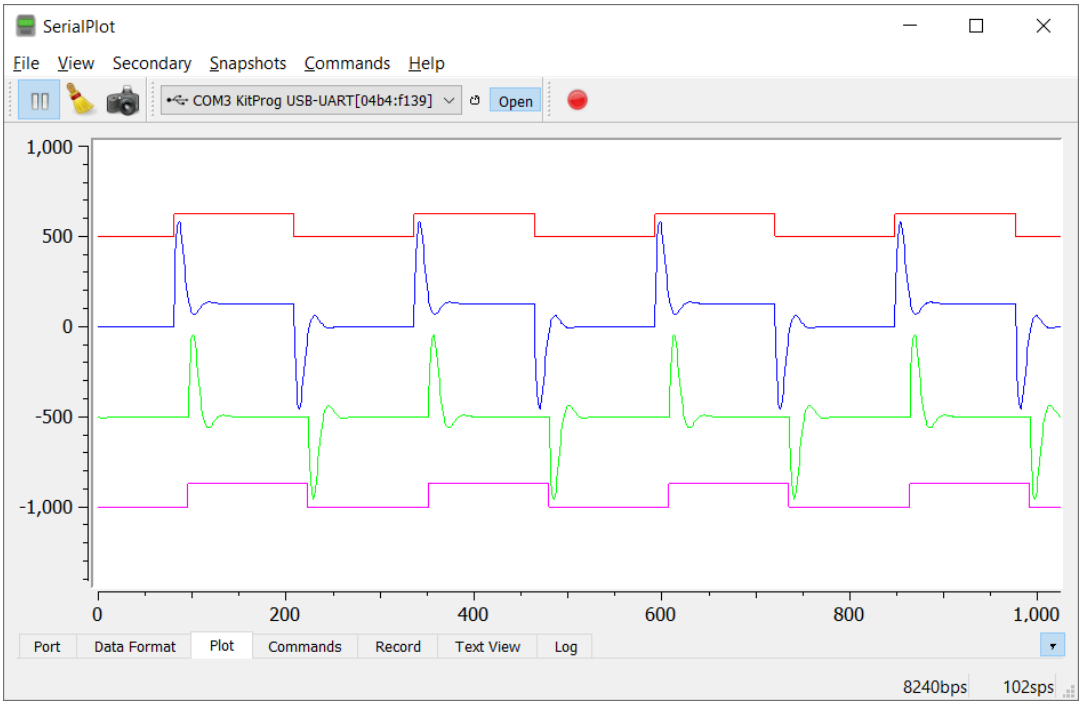


Figure 1. Example of the data stream output by the SerialPlot.

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.

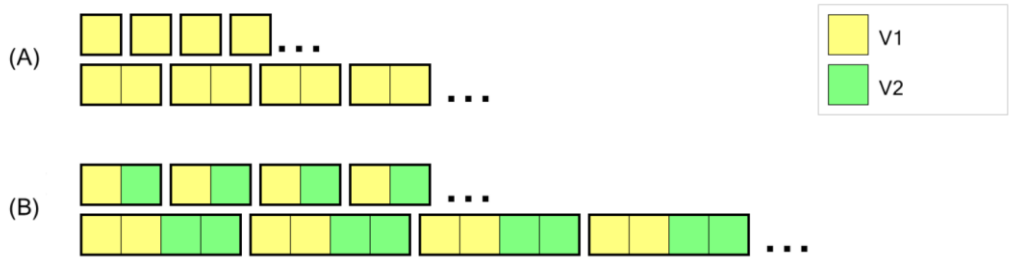


Figure 2. Structure of data packets in the Simple Binary format, each square represents a byte: (A)- single channel of int8 / int16 data type; (B)- 2 channels of int8 / int16 data type.

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** for details.

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 mode of operation is slow (see **Performance** section) due to the intermediate conversion of the data into the human-readable form, and is not recommended. It may be useful, however, for debugging purposes using a character Terminal. See **Appendix 1** for details.

The Custom Frame format

The Custom Frame is the preferred data format for transmitting data. It offers performance comparable to the Single Binary with superior reliability. In this format the data packets are staffed into individual frames, preceded with a frame header[*], and optional checksum byte at the end. Several data packets can be staffed into a single frame, limited by the maximum size of 255 bytes, excluding the header and checksum byte. The SerialPlot charting software shall not update the screen until the entire frame is transmitted. This helps reducing computer CPU load by lowering screen update frequency when data rates are high.

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, selected data type is int16 (2-byte) and number of data packets per frame is 1, the frame structure is:

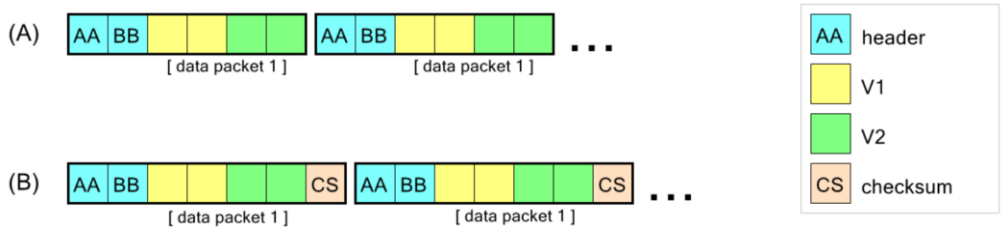


Figure 3. Data stream structure in the Custom Frame format, where each square represents a byte. Header size is 2 bytes, number of channels is 2, data type is int16 and the number of data packets per frame is 1. (A)- checksum disabled, (B)- checksum enabled.

The optional checksum allows protection against data being lost or corrupted. If checksum doesn't match, the entire frame is discarded by the SerialPlot software. See **Appendix 1**, Custom Frame section for details.

The Frame can carry multiple data packets, up to 255[†] bytes total, excluding the header and checksum byte. Adding each data packet to the frame requires its own call to the Chart_Plot() or Chart_PlotArray() procedure. The SerialPlot automatically adds the frames header and checksum according to the settings. Individual data packets shall be transmitted on each Chart_Plot() or Chart_PlotArray() call. However, the SerialPlot software shall not update the screen until the entire frame has been received by computer. This helps reducing the CPU load by lowering the screen update frequency when data rates are high. The structure of the frame carrying several data packets is shown on Figure 4.

[*] Also referred to by the SerialPlot software as "Frame Start".
[†] In the current version of the SerialPlot 0.12.0 the frame size is limited by 255

Figure 4. Frame structure with multiple data packets per frame, where each square represents a byte. Header size is 2 bytes, number of channels is 2, data type is int16 and the number of data packets per frame is 3. (A)- checksum disabled, (B)- checksum enabled.

The maximum number of data packets, that can fit the frame can be calculated as

$$N_{max} = (int)\left(\frac{255}{NumChannels \times DataSize}\right)$$

where DataSize - the size in bytes of the selected data type. For example, in case of 4 channels of data type int16, the maximum number of the data packets is 31. The component performs automatic validation that the parameters selected fit this criterion.

Features not implemented

- Data type *double*
- Variable frame length in Custom Frame format
- Bidirectional communication (Rx + Tx)
- USB-UART communication

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 Chart_Plot() 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 the execution time rises 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 UART speed at 115.2k, compiler release mode with 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

Several demo projects are provided, see **Appendices 1 - 4** 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	
1.0	Number of channels increased to 64. Added multiple data packets per frame. Added procedure Chart_PlotArray(). Added PSoC4 SCB UART support.	

References

1. Serial PLOT v0.12.0, by Hasan Yavuz Özderya,
https://hackaday.io/project/5334-serialplot-realtime-plotting-software/discussion-88924
https://bitbucket.org/hyOzd/serialplot

Appendix 1

Simple Binary format

When data packet fits a single byte (1-channel of int8 or uint8 data type), the component generates the 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(*) copied to the SerialPlot Data Format tab (Figure 5). Recommended size of the complimentary UART Tx buffer in this mode is 4 (hardware FIFO), see **Performance** section for details.

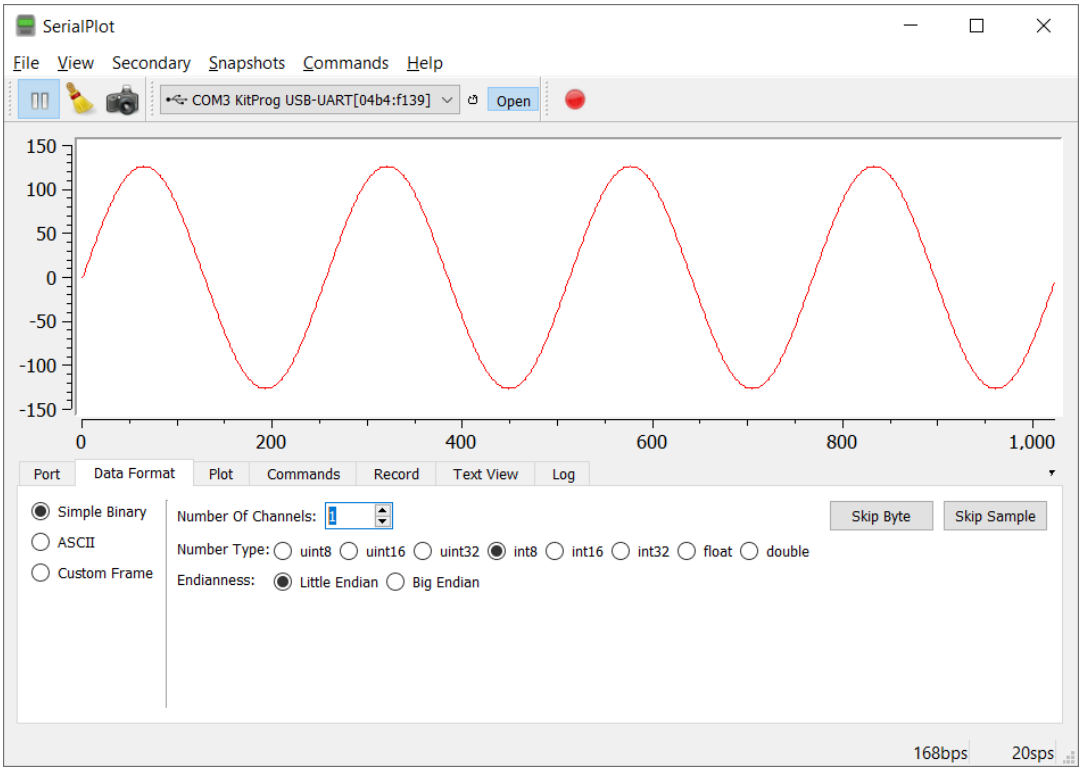


Figure 5. SerialPlot output and data format settings: Simple Binary; Number of Channels: 1; Number Type: int8; Endianness: Little Endian.

* The SerialPlot software doesn't support remote configuration.

When the size of the data packet exceeds one byte (for example 2-channels of int8 data type), the component generates the 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 DataPacket[2] = {V1, V2};

    // send data packet as array of char
    UART_1_PutArray((uint8 *) DataPacket, sizeof(DataPacket));
}
```

Suggested settings should be manually copied to the SerialPlot Data Format tab (Figure 6). 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 (here is 4 bytes). See **Performance** section for details.

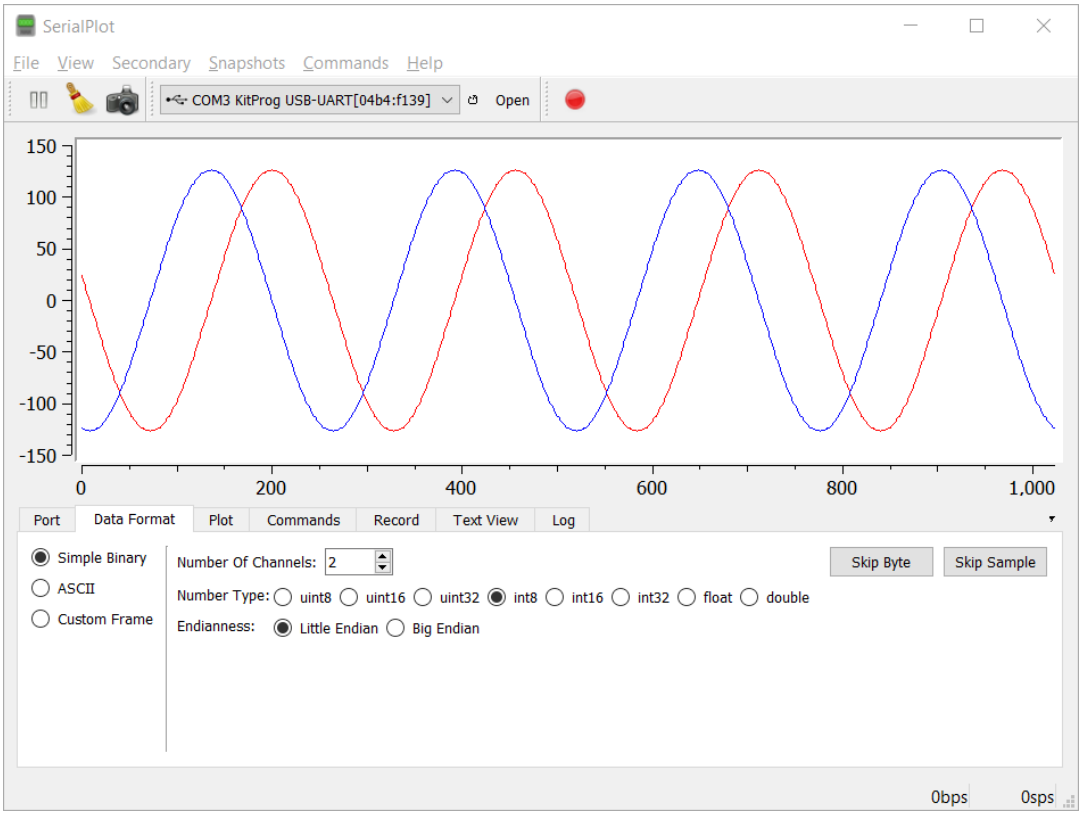


Figure 6. SerialPlot output and data format settings: Simple Binary; Number of Channels: 2; Number Type: int8; Endianness: Little Endian.

ASCII format

Shown below is a code example of sending 2 channels of the float32-type data in ASCII format. Upon compiling the project, based on the Dialog settings, the component generates overload version of the Plot() procedure, along with the suggested SerialPlot settings:

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

    char abuff[36];

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

Suggested settings should be manually copied to the SerialPlot Data Format tab (Figure 7). If float data format selected, the newlib-nano Float Formatting must also be enabled in the Project build settings. The size of the complimentary UART Tx buffer must be set to no less than the size of the abuff[] (here is 36 bytes).

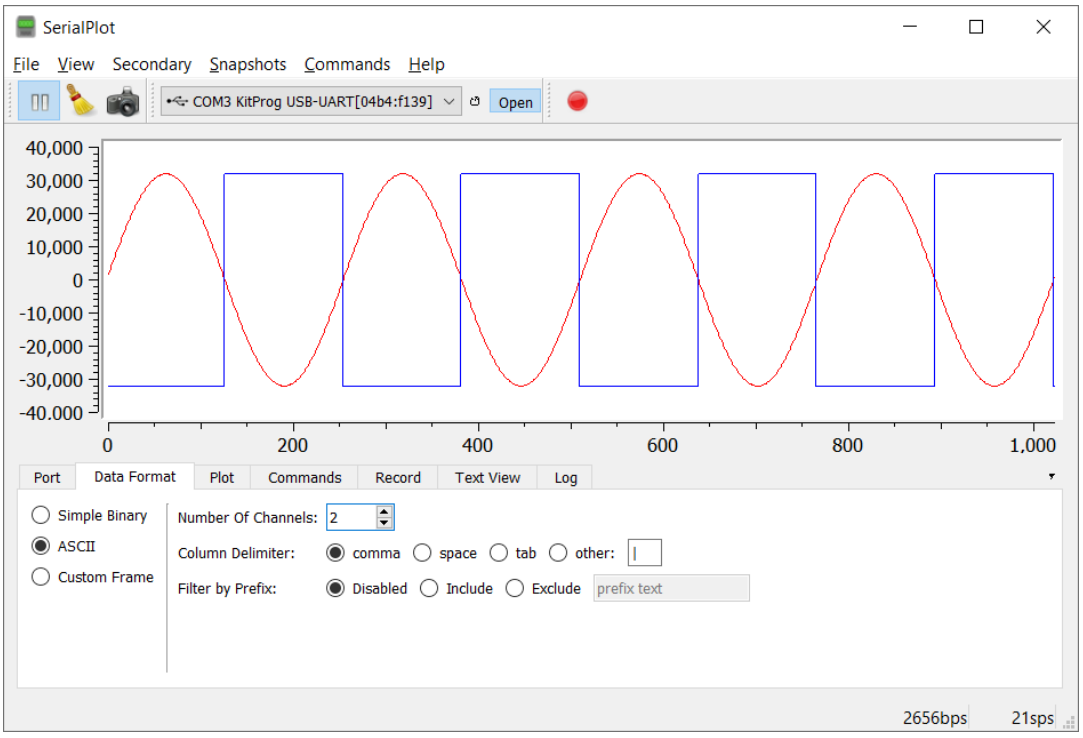


Figure 7. SerialPlot output and data format settings: ASCII data format; Number of Channels: 2; Column Delimiter: comma; Filter by Prefix(*): disabled.

* Filter by prefix is not supported in this version of the component

Custom Frame format

Shown below is a code example of sending 2 channels of the int16-type data in Custom Frame format with Checksum disabled. 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

    static uint16 counter = 0;                      // data packet counter

    uint8  FrameHeader[2] = {0xAA, 0xBB};
    int16  DataPacket[2] = {V1, V2};

    // start Frame by sending a Header
    if (counter == 0) {
        UART_1_PutArray((uint8 *) FrameHeader, sizeof(FrameHeader));
    }

    // send data packet as array of char
    UART_1_PutArray((uint8 *) DataPacket, sizeof(DataPacket));

    if (++counter == 1) {                           // all data packets sent
        counter = 0;                                // reset packet counter
    }
}
```

When Checksum enabled, component generates the overload version of the procedure, which includes 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

    static uint16 counter = 0;                      // frame data packet counter
    static uint8  CS = 0;                           // checksum byte
    uint8 * pCS  = (uint8 *) &CS;                   // pointer to checksum byte

    uint8  FrameHeader[2] = {0xAA, 0xBB};
    int16  DataPacket[2] = {V1, V2};

    // start Frame with a Header
    if (counter == 0) {
```

```
        UART_1_PutArray((uint8 *) FrameHeader, sizeof(FrameHeader));
    }

    // sum all bytes in data packet
    uint8 * pVal = (uint8 *) &DataPacket[0];            // pointer to start byte

    int i;
    for (i=0; i<Chart_1_DataPacketSize; i++)
    {
        * pCS += * pVal++;                              // add bytes to checksum
    }

    // send data packet as array of char
    UART_1_PutArray((uint8 *) DataPacket, sizeof(DataPacket));

    if (++counter == 1) {                               // all data packets sent
       UART_1_PutChar(CS);                              // send check sum
        counter = 0;                                    // reset packet counter
        CS      = 0;                                    // reset checksum
    }
}
```

Settings provided in the comments header should be manually copied to the SerialPlot Data Format tab (Figure 8). The size of the complimentary UART Tx buffer in this mode should be set to no less than the frame size.

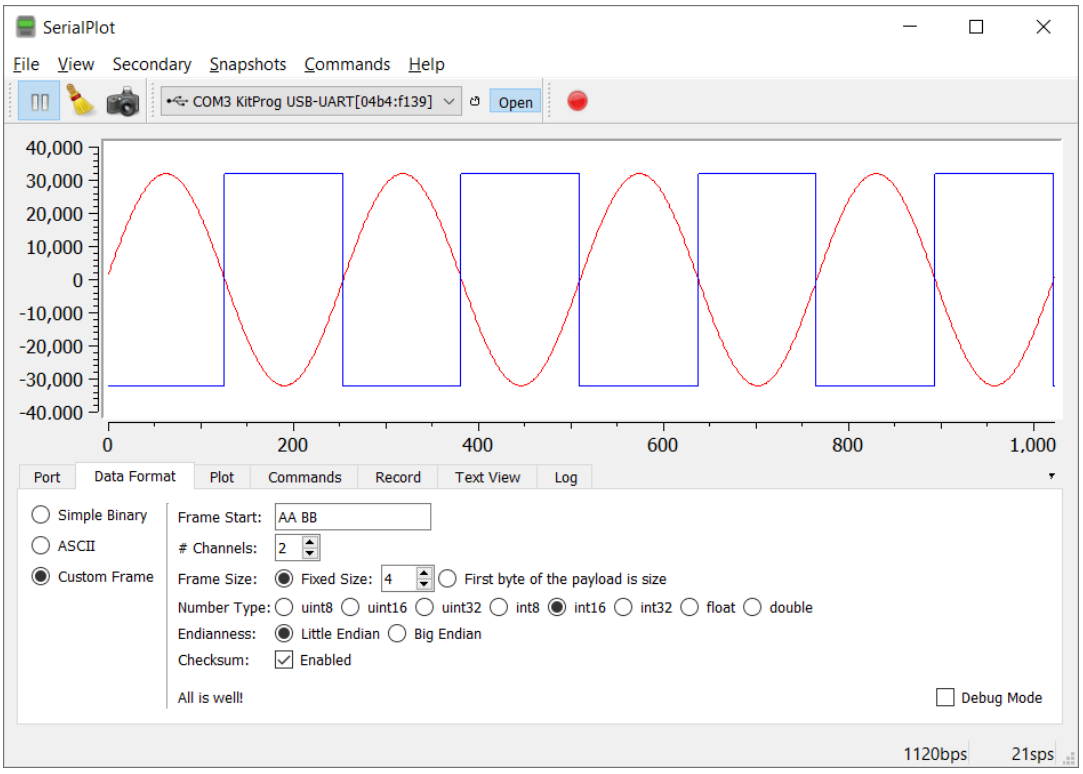


Figure 8. SerialPlot output and settings: Custom Frame mode; Frame Start: "0xAA, 0xBB"; #Channels: 2; Frame Size: Fixed, Size=4; Number Type: int16; Checksum: Enabled.

Appendix 2

PSoC5 basic demo

The PSoC5 project schematic using SerialPlot component is shown on Figure 9. Test data samples are generated on clock timer and streamed to the host computer using (UDB) UART through USB-UART bridge, built into the KitProg.

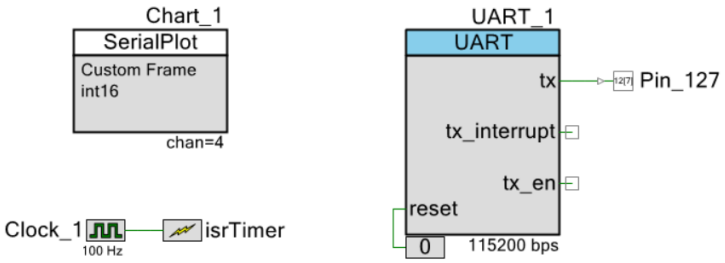


Figure 9. Project schematic.

The test data are generated on 100 Hz timer clock. The Chart component is configured for Custom Frame format, 4 channels, int16 data type. UART_1 Tx buffer size is 128 bytes. The SerialPlot software viewport and configuration page are shown on Figure 10.

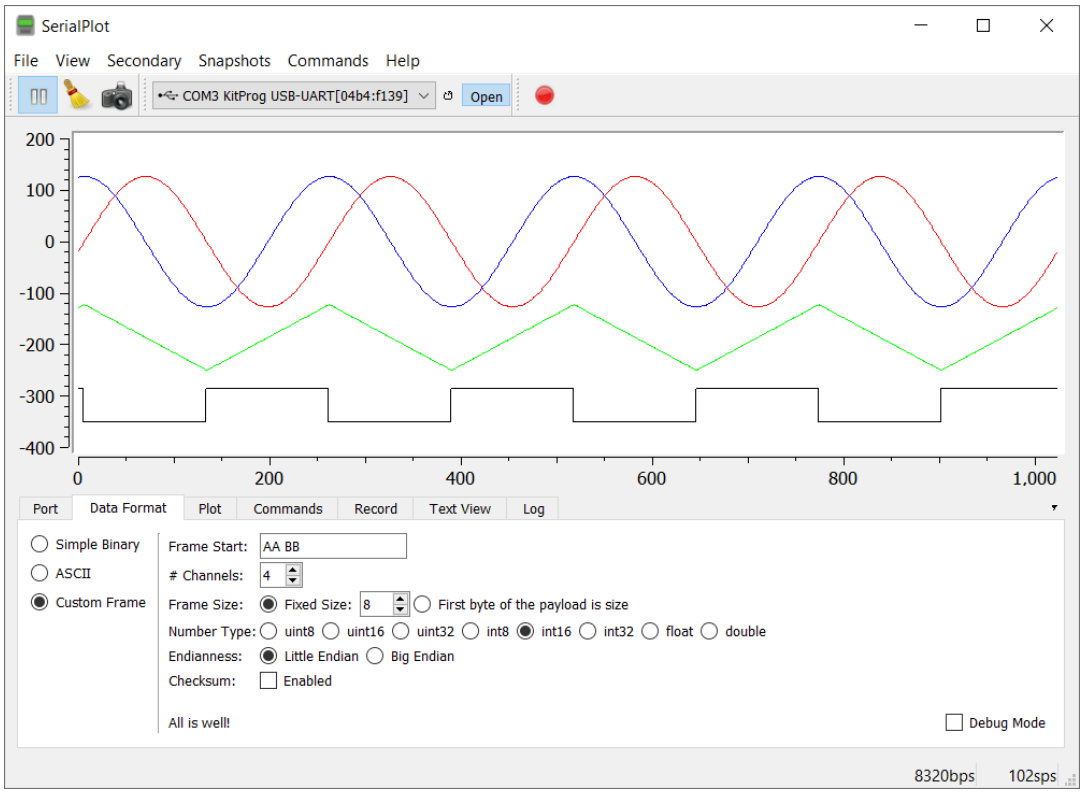


Figure 10. SerialPlot output and settings: Custom Frame mode; Frame Start: "0xAA, 0xBB"; #Channels: 4; Frame Size: Fixed, Size=8; Number Type: int16; Checksum: Disabled.

Project essential code:

```
int16 v1,v2,v3,v4;                          // channels data

for(;;)
{
    if(isrTimer_flag != 0)                  // monitor for Timer interrupt
    {
        isrTimer_flag = 0;                  // reset flag

        GenerateSignal();                   // update values v1-v4

        Chart_1_Plot(v1, v2, v3, v4);       // plot values
    }
}
```

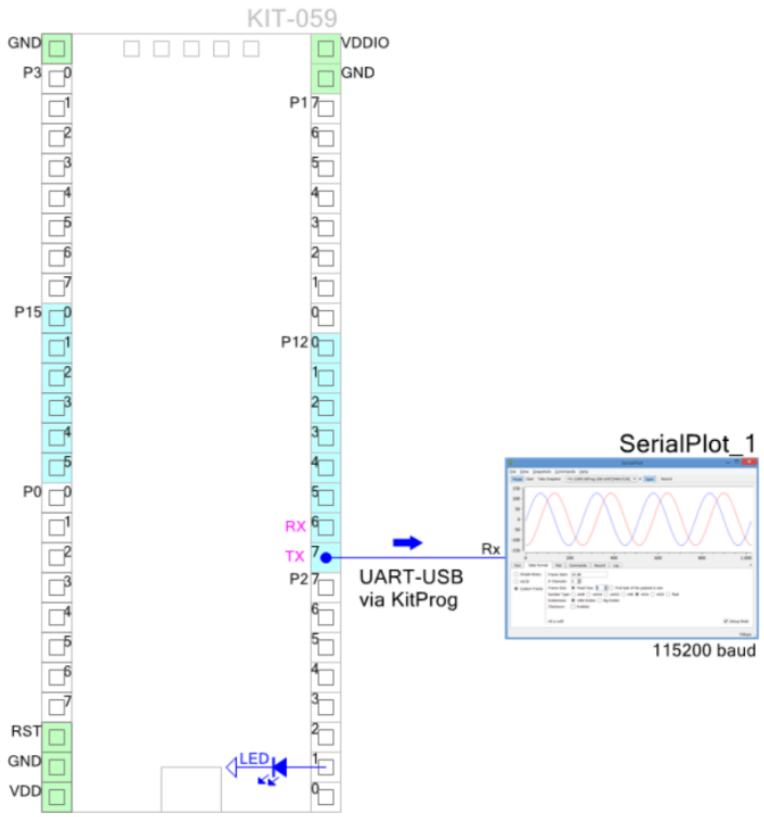


Figure 11. Project annotation for PSoC5 CY8CKIT-059 using PSoC Annotation library v1.0.

Appendix 3

Array plotting demo

The PSoC5 ArrayPlot API demo is shown on Figure 12. Test data samples are generated on clock timer and streamed to the host computer by UART_1 using the USB-UART bridge built into the KitProg.

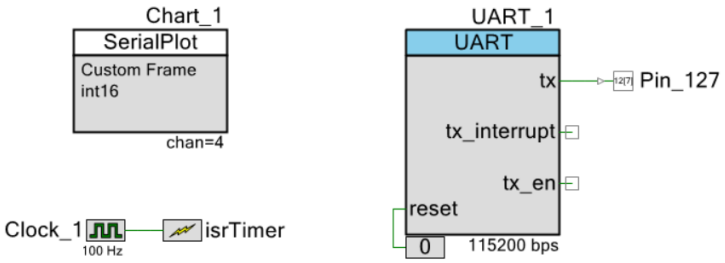


Figure 12. SerialPlot demo project schematic.

The test data are generated on 20 Hz timer clock. The Chart component is configured for Custom Frame format, 32 channels, uint8 data type, checksum enabled. UART is configured for Tx-only, buffer size 128 bytes, speed 115k.

Sample code of sending 32 channels of data at once:

```
uint8 Channels[32];                         // array of data channels

for(;;)
{
    if(isrTimer_flag != 0)                  // monitor for Timer interrupt
    {
        isrTimer_flag = 0;                  // reset flag

        GenerateSignal();                   // update Channels[] array

        Chart_1_PlotArray(Channnels);       // plot array of channels
    }
}
```

The SerialPlot viewport and settings tab are shown on Figure 13. The SerialPlot viewport is configured for bar plot display. The SerialPlot settings a shown on the Data Format tab; they should match the Chart component parameters (Custom Frame, 32 channels, uint8 data type, checksum enabled). The length of the data frame is set to 32 bytes (32 channels x 1 byte). This settings can be found on the component popup window, and manually copied to the SerialPlot Data Format tab, Figure 14.

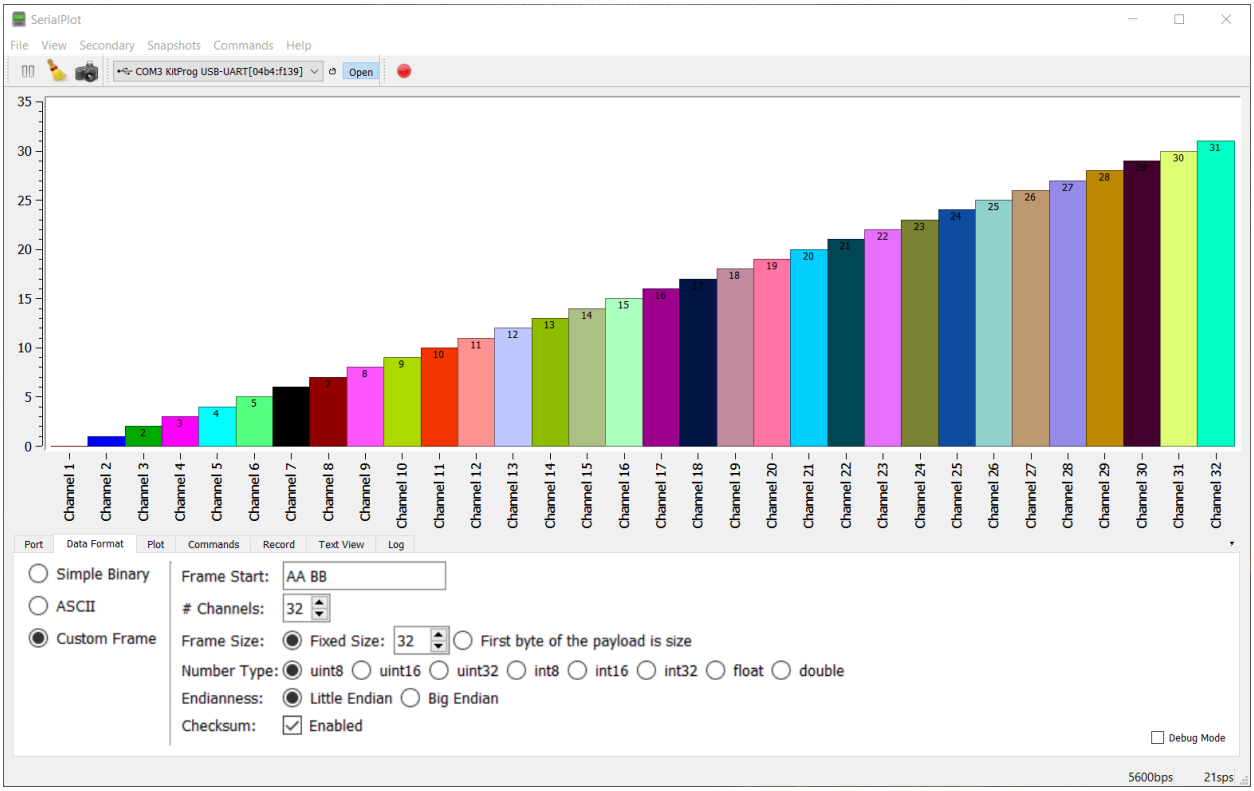


Figure 13. SerialPlot bar plot output and settings: Custom Frame mode; Frame Start: "0xAA, 0xBB"; #Channels: 32; Frame Size: Fixed, Size=32; Number Type: uint8; Checksum: Enabled.

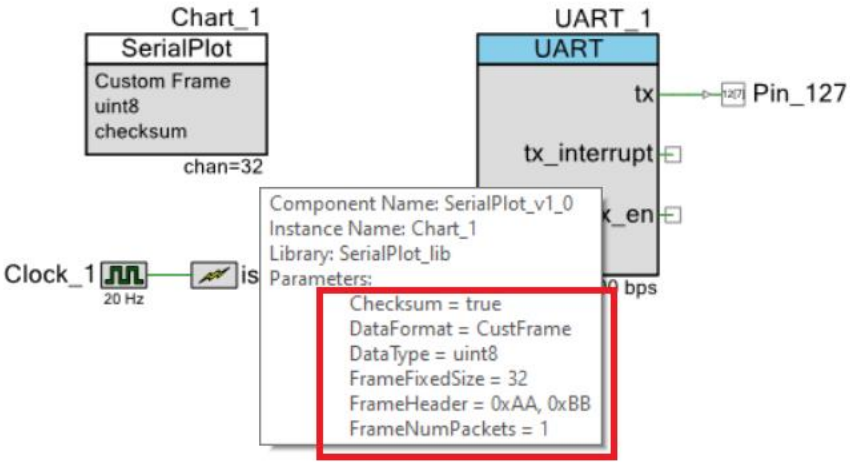


Figure 14. Component Popup window shows required SerialPlot settings.

Appendix 4

PSoC4 SCB UART demo

The PSoC4 project schematic using SerialPlot and SCB UART is shown on Figure 15. The data samples are generated on periodic WDT interrupt and streamed to the host computer using SCB UART through the USB-UART bridge, built into the CY8CKIT-044 Pioneer Kit.

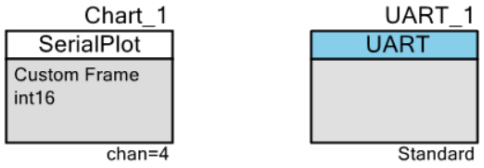


Figure 15. PSoC4 SCB UART project schematic.

The test data are generated on 125 Hz WDT Timer0. The Chart component is configured for Custom Frame format, 4 channels, int16 data type. The SCB UART is configured for Tx only, 115k rate, Tx buffer size set to 128 bytes. The UART output is assigned to the hidden Pin_7[1], which is directly connected to the onboard USB-UART bridge on the CY8CKIT-044 board (Figure 17). The SerialPlot software viewport and configuration page are shown on Figure 16.

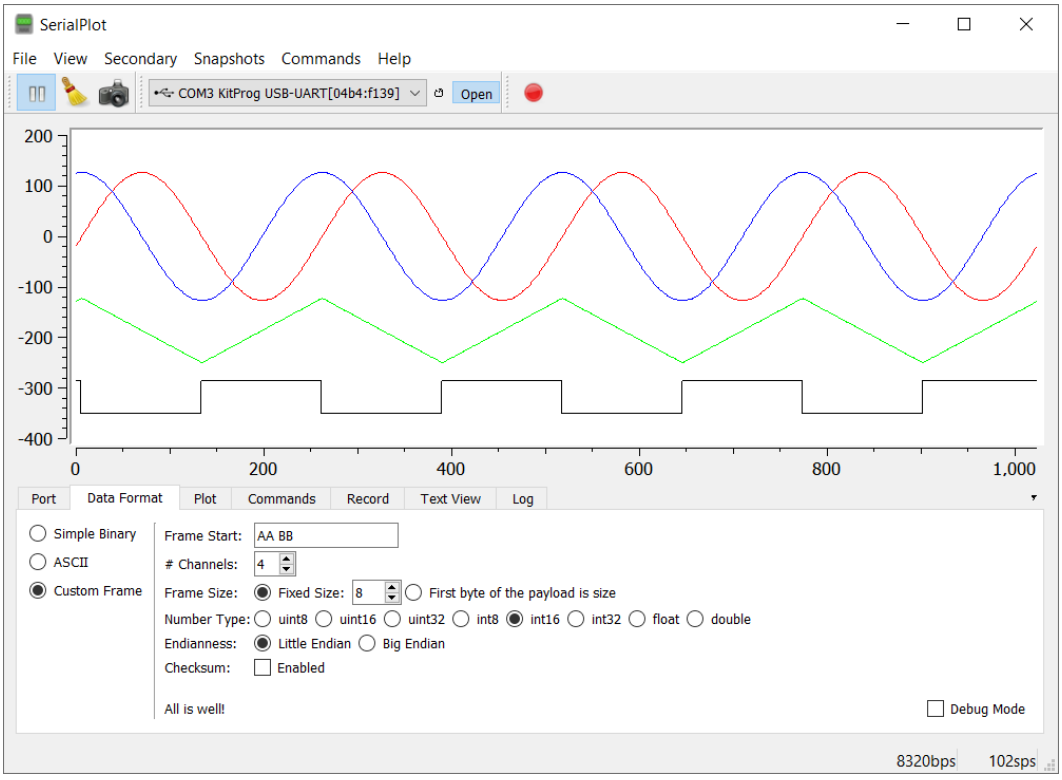


Figure 16. SerialPlot output and settings: Custom Frame mode; Frame Start: "0xAA, 0xBB"; #Channels: 4; Frame Size: Fixed, Size=4; Number Type: int16; Checksum Enabled: False.

Project essential code:

```
int16 v1,v2,v3,v4;                          // array of data channels

for(;;)
{
    if(isrTimer_fla != 0)                   // monitor for Timer interrupt
    {
        isrTimer_flag = 0;                  // reset flag

        GenerateSignal();                   // update values v1-v4

        Chart_1_Plot(v1, v2, v3, v4);       // plot values
    }
}
```

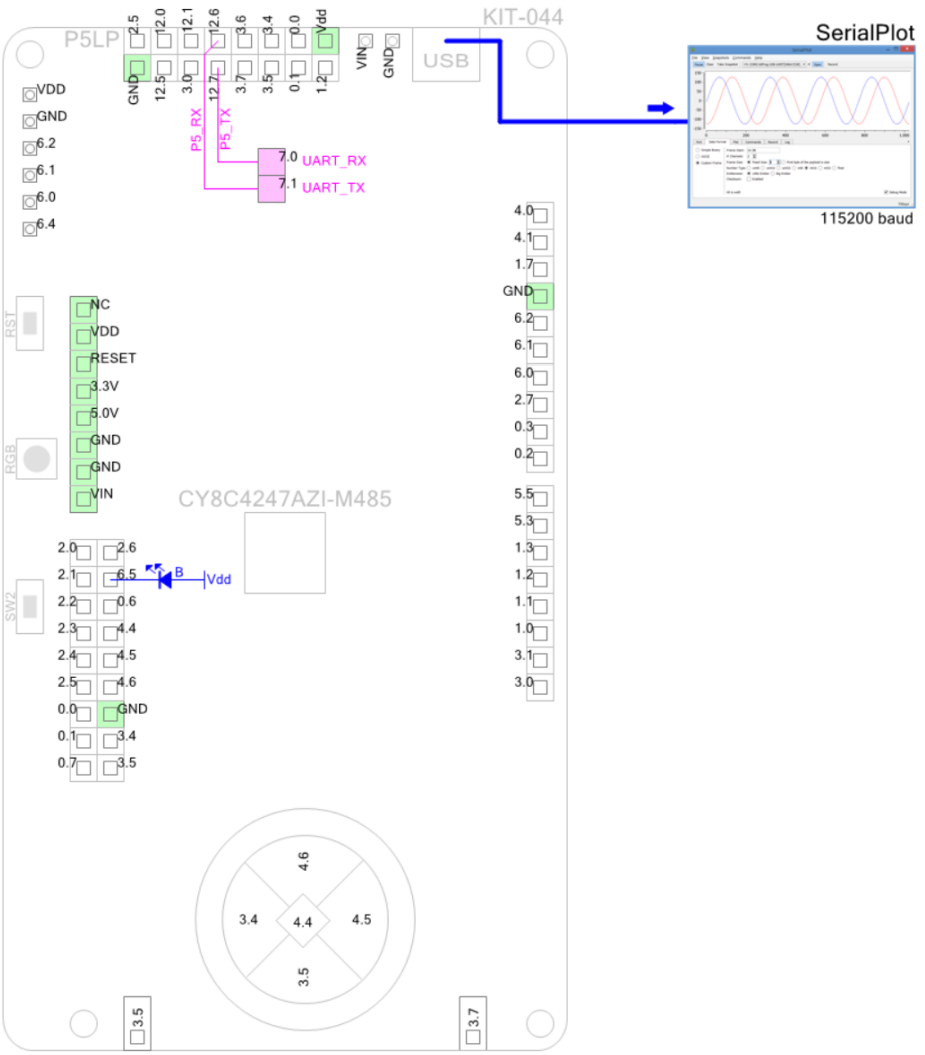


Figure 17. Project annotation using [KIT-044 v0.2 annotation component]. The hidden SCB UART Tx Pin_7[1], directly connected to the onboard USB-UART bridge, is shown in pink.