GreenSocs®

Serial Socket

Document History

Date	Author	Comments	Version
10 March 2010	Manish Aggarwal	Initial Draft	0.1

Table of Contents

1 Purpose

The purpose of this document is to explain the design and implementation of the serial socket project.

2 Motivation

DisplaySocket is a GreenSocket based TLM-2.0 graphics socket. It carries the same advantages as other GreenSocket based sockets, but caters specifically for graphical displays.

To create the model of a SoC/IP that deals with image processing, it becomes necessary to connect the model with a display, to show the results.

The DisplaySocket framework provides an interface specifically designed for the efficient connection of display driver models to the user’s host simulation environment, or to verification components designed to check the functionality of the display controller model.

We refer to the object that finally displays the image in the user’s host simulation environment, or performs the verification as a ‘back end’.

The DisplaySocket framekwork comes complete ‘backe-ends’ designed for use with X11 based simulation environments. It also contains a GreenRouter based interconnect that can be used to connect many display drivers to many ‘back-ends’.

Why one should use this project? What are advantages of using it instead of using other alternative projects available.

3 Overview

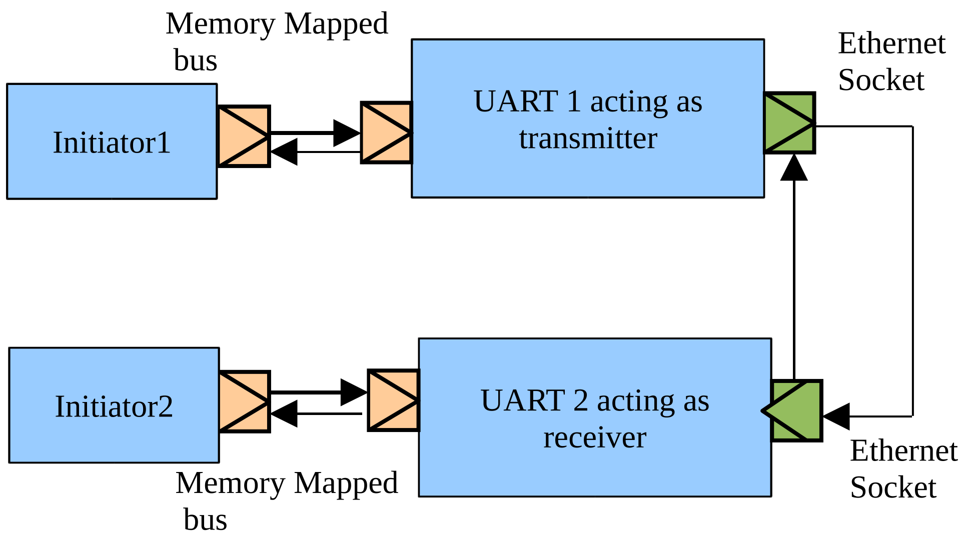
3.1 Background

Please refer GS_Serial.pdf document located at https://svn.greensocs.com/public/packages/serialsocket/docs.

3.2 Related Projects

GreenSocket
https://svn.greensocs.com/public/packages/greensocket

4 Architecture Overview



In the above model InitiatorX is connected to the UART using TLM based memory mapped based. Initiator can configure, initialize and can read/write on the register interface of UART using this bus. The connection between two UART is through the serial socket, whose description is given below.

5 User Guide

5.1 Install Instruction

- Get the latest source from the svn location https://svn.greensocs.com/public/packages/serialsocket
- In the include directory under serial socket lies the code for the serial initiator and target socket.

To run the example:

- View example/Makefile.defs file. It uses four environment variable to know where the required packages are located in the system.

- Use export command to set them:

```
export SYSTEMC_HOME=<systemc path>
export TLM_HOME=<tlm path>
export SERIALSOCKET=<Path to serial socket>/include
export GREENSOCKET=<>directory path where greensocket resided>
```

- Go to example/simple directory. Run 'make'.
- If everything goes fine, you see main.exe appearing in the directory.
- Run main.exe to execute the program.
- See master.h and slave.h file to see how the serial socket are being used.

5.2 Reference Manual

5.2.1 Basics

TLM Serial Payload

The fields in the serial payload are as follows:

Fields	type	Description
`m_command`	enum	There are 2 command supported for AsyncSerial payload transmission : `SERIAL_SEND_CHAR_COMMAND, SERIAL_BREAK_CHAR_COMMAND.`
`m_data`	uint8_t *	The pointer to data array. Each element is of type unsigned char which represents the data bits of a character sent in the Asynchronous Serial transmission.
`m_length`	uint32_t	Number of characters in the data arary
`m_response_status`	enum	Response status, when a module receives the packet(default : `SERIAL_OK_RESPONSE).`
`m_enable_bits`	uint16_t	Each bit in this field is to enable an option as defined below.
`m_parity_bit`	Bool*	Applicable only if m_parity_enable is true. Array of bools. Length of array equals to m_length. Each element specify the parity bit corresponding to each character in m_data field. Also this field is valid only if `m_enable_bits[0]` is set.
`m_valid_bits`	uint8_t	Number of valid bits in each character data sent. Should be between 5-8, depending on the register settings. In case this

		fields get mismatched with received frame then controller can ignore the received packet.
`m_stop_bits`	uint8_t	Number of stop bits in the end of a character. Permissible values are 1, 2 or 3 (3 represents 1.5 stop bits). In case this fields get mismatched with received frame then controller can ignore the received packet or generates Frame Error at target end.
`m_baudrate`	uint32_t	The baud rate at which the communication is taking place. So in case of target baud-rate is different this value then target can discard all further incoming data till the baud-rate values matched again.

Field Values:

m_response_status is of type serial_response_status (enum)

```
      enum serial_response_status {
          SERIAL_OK_RESPONSE = 1,
          SERIAL_ERROR_RESPONSE = 0,
      };
```

Note:- Here default response is `SERIAL_OK_RESPONSE`, where it is recommended that target should not change this response. Since in serial link there are no acknowledgement shared between 2 communication blocks and if target changes the response to `SERIAL_ERROR_RESPONSE` which indicates that serial link will break. And no further communication will happen.

serial_response_status fields	Explanation
`SERIAL_OK_RESPONSE`	If the transaction is successfully delivered (without any errors).
`SERIAL_ERROR_RESPONSE`	If any other error occurred then error response will be generated which will cause the break of serial link and no further communication will happen.

m_enable_bits is to enable some options as explained in the table below:

Bit No	Explanation
0	To enable the parity. Field m_parity_bit is valid if this bit is set to 1.
1	Field m_valid_bits is valid if this bit is set to 1.
2	Field m_stop_bits is valid if this bit is set to 1.
3	Field m_baudrate valid if this bit is set to 1.

For convenience following enum is defined:
```
enum serial_enable_flags{
 SERIAL_PARITY_ENABLED=0x1,
 SERIAL_VALID_BITS_ENABLED=0x2,
 SERIAL_STOP_BITS_ENABLED=0x4,
 SERIAL_BAUD_RATE_ENABLED=0x8
};
```

m_command is enum type, which state type of payload send for communication :
```
enum serial_command {
     SERIAL_SEND_CHAR_COMMAND,
     SERIAL_BREAK_CHAR_COMMAND
};
```

Getter/Setter APIs Provided by the SerialSocket

Following are the getter/setter functions provided by serial socket.

API	Description
`serial_command  get_command()`	Returns the command set in the payload.
`void set_command(const serial_command command)`	Sets the command set in the payload.
`unsigned char*  get_data_ptr()`	Gets the data pointer set in the payload.
`void set_data_ptr(unsigned char* data)`	Sets the data pointer in the payload with the required value.
`unsigned int get_data_length()`	Gets the data length.
`void set_data_length(const unsigned`	Sets the data length.

`int length)`	
`bool is_response_ok()`	Returns true if the response is `SERIAL_OK_RESPONSE`
`bool is_response_error()`	Returns true if the response is `SERIAL_ERROR_RESPONSE`
`serial_response_status get_response_status()`	Gets the response status. The value should be from `serial_response_status` enum.
`void set_response_status(const serial_response_status response_status)`	Sets the response status. The value should be from `serial_response_status` enum.
`std::string get_response_string()`	Get the string for response status defined in the `serial_response_status` enum.
`unsigned char get_num_stop_bits_in_end()`	Get the number of stop bits set that must be followed with each character transmitted..
`void set_num_stop_bits_in_end(const unsigned char stop_bits)`	Set the number of stop bits that must be followed with each character transmitted.
`unsigned char get_valid_bits()`	Get number of valid bits each character is composed of. The value should be between 5-8.
`void set_valid_bits(const unsigned char valid_bits)`	Set number of valid bits ineach character sent.
`uint16_t`	Get the value of m_enable_bits. It tells us which all (Valid, Parity, Stop and Baud Rate) options are

`get_enable_bits()`	enabled)
`void set_enable_bits(uint16_t enable_bits)`	Set the value of m_enable_bits.
`bool isParityEnable()`	Returns true if parity bits option is enabled.
`bool isValidBitEnable()`	Returns true if valid bits option is enabled. If it is false then the valid bits in a char is assumed to be 8.
`bool isStopBitEnable()`	Returns true if stop bits option is enabled.
`bool isBaudRateEnable()`	Returns true if Baud rate is set.
`bool* get_parity_bits() const`	If parity bits option is enabled, it returns the pointer to array which contains the parity for each character sent. Number of elements in the array returned is equal to the value of the data length.
`void set_parity_bits(bool* parity_bits)`	Sets the pointer of an array of bools containing the parity bits for each character.
`unsigned int    get_baudrate()`	Get the current baud rate value. It can be used to calculate the value of delay that should be realized at the receiver side or the rime after which the receiver will get the characters.
`void set_baudrate(unsigned int baudrate)`	Set the current baud rate value.

6 Example Walkthrough

In this example the simple example present in the example director is explained.

It consist of three files:

- Top level file is main.cpp which instantiates master and slave modules and binds them.
- In master.h file, the main thread run() creates a serial transaction, sets the various fields like command, data pointer, data length, number of stop bits, parity bits and other fields and send it to the slave.
- Slave registers the b_transport, receives the transaction, decodes the fields in the payload and show it on the screen.