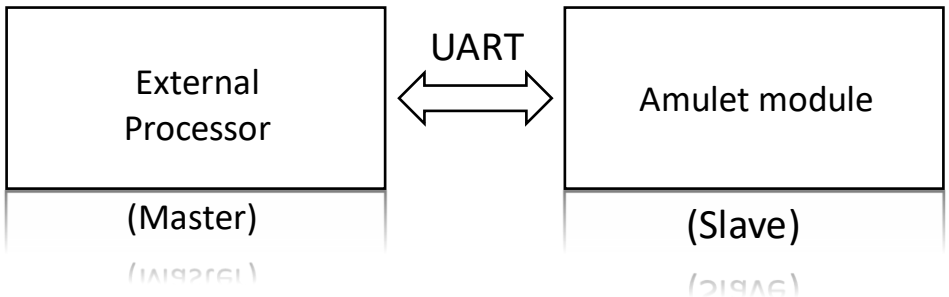
Amulet UART Overview

UART is a common interface for communicating with an Amulet module. In order to easily communicate with the module, Amulet offers two command protocols that expose the features of the Amulet module: one is ASCII-based and the other is CRC-based. The CRC-based protocol has a similar structure to Modbus RTU. There are two UART ports on the board; known as COMM and PROG, which are used for both USB and UART or dual UART/USB is an option. The UART supports up to 115200kpbs and connects like a standard RS-232. Within the UART module the Amulet is able to support both Master and Slave UART communications.

The Graphical OS that Amulet developed handles all UART communications in a serial manner, and can send and receive master/slave commands asynchronously without any efforts from the user. For master mode, GEMstudio exposes two methods of making function calls: one is directly from Control Widgets in GEMstudio and the other is from GEMscript; GEMscript is a programming language embedded inside of GEMstudio designed to give extra flexibility.

The Amulet module uses the onboard RAM as the UART communication buffer. There are over 1kilobits of onboard RAM that has been allocated as Virtual Dual Port RAM (VDP-RAM) for this buffer. This VDP-RAM is accessible from the Amulet Widgets and can also be stored in flash for a level of permanency. This VDP-RAM is structured as variables and internally accessible through the InternalRAM structure. There are 256 different byte variables, 256 word (16-bit) variables, 256 color (32-bit) variables, and 256 25-character null terminated string variables (25+1=26 byte allocated per string variable). All of these variables are directly addressable through the UART, and can be changed singularly or through array reads and writes.

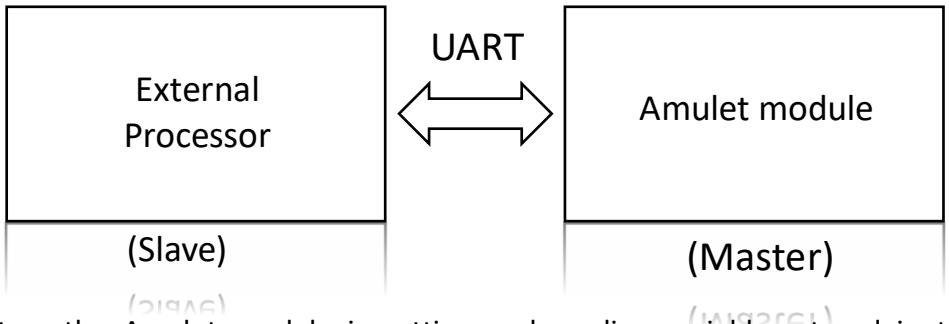
UART Slave Mode Functionality



In this system the external processor is the master and is setting and reading the InternalRAM variables through the VDP-RAM in the module. For the GEMstudio side, nothing need to be done to for the module to accept slave commands, all of the work is done in the Amulet Graphical OS. An example of the module in slave mode would be updating data in a graph in real-time. Inside GEMstudio the InternalRAM variables woulc be used in View Widgets for updating the data array used in graphing. This means as the external master processor is setting data into the VDP-RAM, the graph on the display will be updating automatically. If more flexibility is required this data can also be accessed directly from GEMscript, so any data processing that is required can done in code.

On the external processor side, an implementation of the Amulet master protocol is required. These commands perform basic set and read actions on specific variables or groups of variable types, such as an array of bytes or an array of colors. The appropriate protocol format, ASCII or the CRC, functions are required.

UART Master Mode Functionality



In this system the Amulet module is setting and reading variables stored in the external processor. For the Amulet module to operate in master mode, the slave processor needs a memory space allocated that maps the VDP-RAM. The slave protocol is straightforward; a code example of the CRC protocol is available in C++.

When communicating in master mode from the Amulet module, there are two methods of initiating the Amulet:UART commands in GEMstudio. One is through the use of an href inside of Widgets, and the other method is from GEMscript. Initiating a UART command from a View Widget could is as simple as String Field with Amulet:UART.byte(0x00).value() in its href. This would get the value stored in byte address 0x00 from the external processor and display that value on the screen. More intricate UART communications can be implemented through GEMscript, allowing the two systems to efficiently communicate.

When in master mode reading data from the slave processor is essentially taking the data that is stored in the memory space of the external processor and mapping it over the InternalRAM variables. The external slave processor must support any command the Amulet module will be sending out.

Command Format Example (CRC protocol specific)

When using the UART port, either in master or slave mode, the commands will look the same. If the external processor is the master the Get Byte command is going to look the same as if the Amulet Module is sending that command to the external processor. The command in GEMstudio looks like “Amulet:UART.byte(x).value()”. This was used in the last example. This is a GET Byte request. It will send a command to the slave external processor and if the command is value, the slave processor should respond with the value stored in byte address location “x”.

In the CRC protocol there is a 16-bit CRC calculation which is appended to the end of each command. This CRC is calculated by running the command through a CRC algorithm, which will create a unique number. The slave processor has to take the message, run the command through a CRC algorithm, and compare the two CRC values, the one calculated and the one transmitted.

If they are the same the message was sent without corruption, if it fails the CRC check something is wrong with the command. There is no such data validation done in the ASCII protocol.

The commands being sent have a specific custom format. For example in the CRC protocol, if the GEMstudio file being compiled has a view widget with an href of Amulet:UART.byte(0x1A).value(), which will send out the "Get Byte Variable #0x1A" request, the message to be transmitted would consist of five bytes. The first byte is the Host ID (0x02), the second byte is the "Get Byte Variable" opcode (0x20), third byte is the byte variable number (0x1A), fourth byte is the LSByte of the 16-bit CRC of the first three bytes (0x48), and the seventh and final byte is the MSbyte of the 16-bit CRC (0x0B). So the five byte message looks like: 0x02 0x20 0xA1 0x48 0x0B

In this example above if the external processor is the master, it would need to be able to generate, and send out the 5-bytes above over a serial connection, wait for the slave to either acknowledge (ACK) or negatively acknowledge (NACK) the command, and then read the response off the serial port from the slave with the value. Ever command being sent and every response back has a specific format and are easy to implement.

As stated earlier before, the Amulet Graphical OS is capable of performing both master and slave protocols synchronously, it does not need to be configured into one mode or the either. A full master/slave external processor is very powerful in enabling a bi-directionality communications environment.