Using the GE Reflective Memory DMA Custom Device

This custom device is not officially supported through normal NI support channels. For any support issues that arise, please submit an issue on github. This code is subject to the license terms found there.

# Theory of Operation

Reflective memory cards, at the driver level, behave just like memory (RAM). There is a set of memory addresses, and whenever their contents are changed the cards update their peers behind the scenes. This updating process is normally well abstracted from the user unless an advanced library is used.

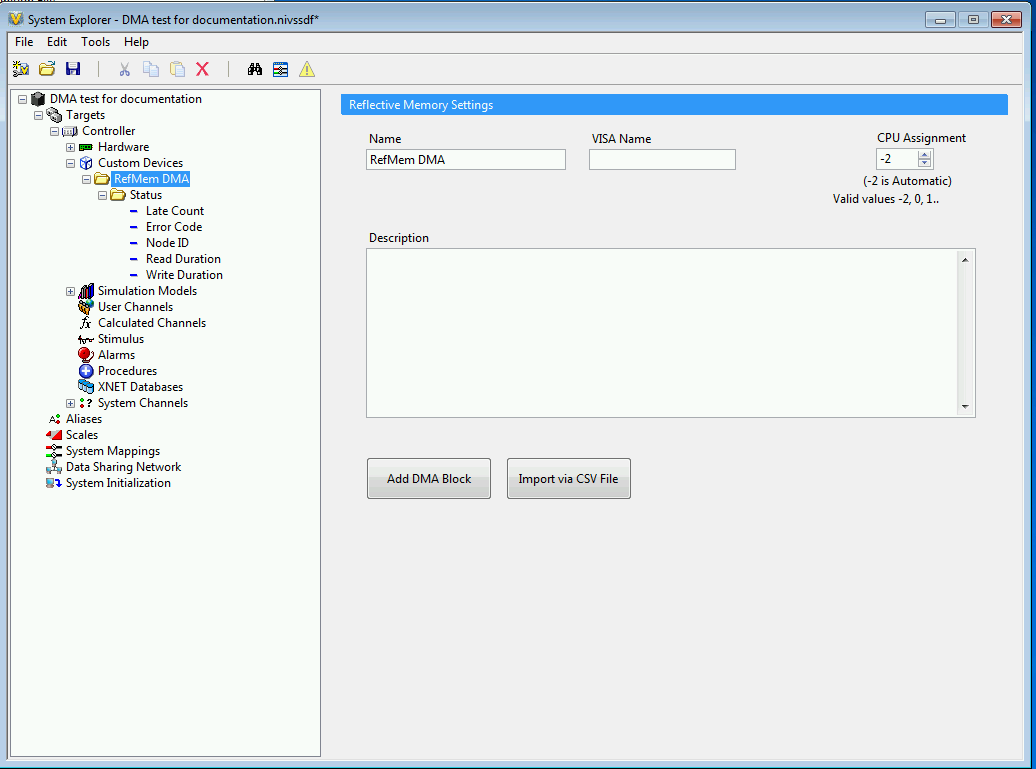
VeriStand brings the constraint to the table that it must have its channels in Double format. It does that because doubles have the precision necessary for simulation and modeling. VeriStand also hides these details from users – more on that in the Coming from VeriStand section below.

The built-in Data Sharing reflective memory feature in VeriStand uses the PIO model of access to communicate with the card. Essentially, it reads and writes registers for individual access of the data channels. Each channel takes an additional exchange with the card. This is optimal for very low channel counts, as they are small interactions. It is also optimal for memory maps that are not contiguous. But, it does not scale well, and for large channel counts can be cumbersome for the machine as it consumes CPU time for each interaction.

This custom device, by contrast, is built on the DMA (Direct Memory Access) interface. It is able to buffer many channels at once for a read or write, for much higher throughput (at high channel counts, at least). This comes with the constraint, however, that channels in each buffer must be contiguous in the memory map. Non-sequential addresses can’t be written to in one DMA interaction. For the sake of using this custom device, those sequential sets of memory addresses are called blocks.

# Main page

On the main page, there are a few configuration options:



* **Name:** the user may provide a name
* **VISA name:** this must correspond to the VISA alias of the desired GE reflective memory card. That can be found by consulting Measurement and Automation Explorer (MAX). There you can look up the VISA alias of the hardware, change it, and more.
* **CPU assignment:** The default is -2, or automatic. VeriStand is generally very good about automatically assigning cores. It is probably best to leave this as is unless you have a specific reason to change it.
* **Description:** The user can enter a description here, to help document their VeriStand project.
* **Add DMA Block:** Adds a DMA block. See the Adding a Block section below
* **Import via CSV file**: imports from CSV file. See the section below.

# Importing via CSV file

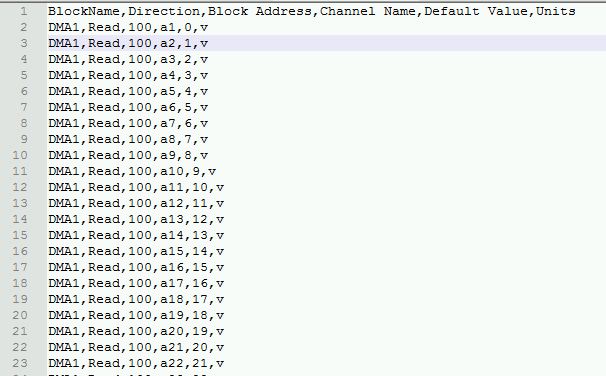
This is a quick way of configuring the system. Many customers find that managing CSV files is easier than any GUI. They can be easily modified, backed up, and version controlled.

This button will import a CSV file and from it create the appropriate DMA blocks and channels.

There is a sample CSV file in the Examples folder.

# CSV file format

For now at least, the format is as follows:

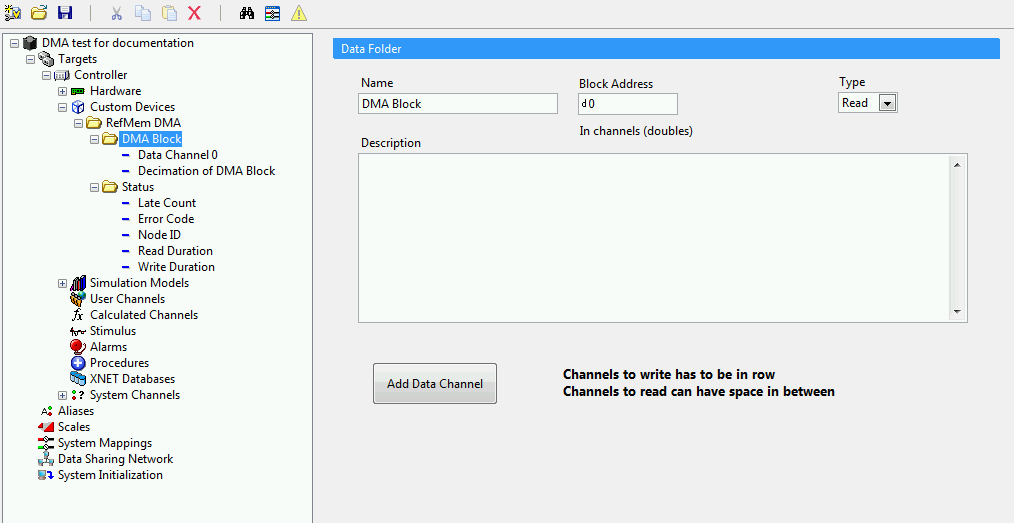


# Adding and configuring a Block

When the Add Block button is pressed, a new Block is created.

A Block represents a set of data channels that lie adjacent to each other in the memory table. A non-sequential set of memory addresses will need to get broken up into multiple blocks so that each is sequential.

A Block can be configured on its page:

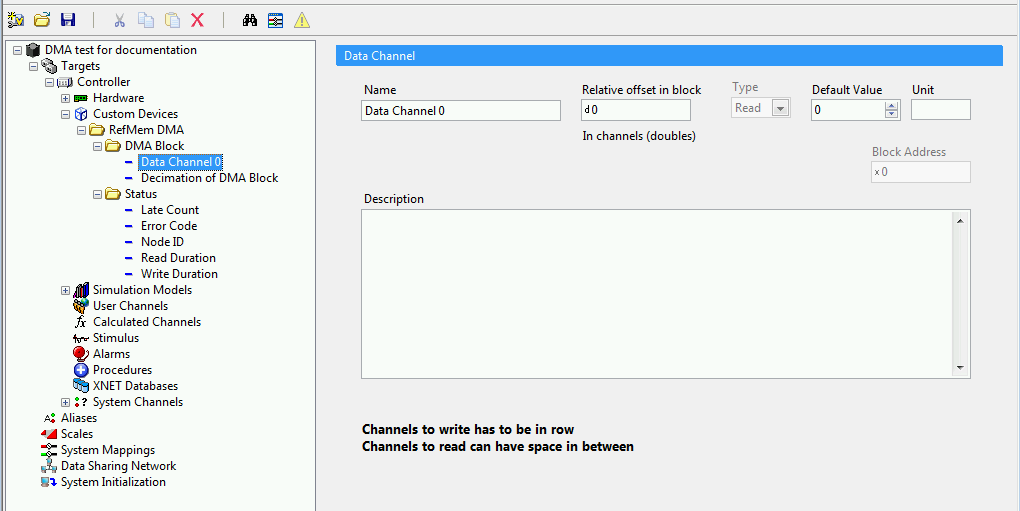


* **Name:** The user can name the block. Naming them appropriately will help keep track of them – you might want to put “read” or “write” in the name, to help keep them straight.
* **Block Address:** Here, the starting memory address of the block is set. There is a catch: since VeriStand uses doubles, you would have to take the number of desired (double) channels and multiply by 8. That is because doubles are 8 bytes in length. By contrast, memory is addressed per byte. Here, this address value is (memory address \* 8), or it is in terms of doubles. If you have another system you wish to use the same data table, multiply this value by 8 in order to find the raw memory address for that system to use.
* **Type:** read or write. This applies to the whole block.
* **Description:** You can add a description.
* **Add Data channel:** see that section below
* **Decimation of DMA block:** This is a setting that is in a channel, since it can be changed at runtime. When set to 0 (the default), it is disabled, or every channel will be read on each iteration of VeriStand’s primary control loop (PCL). When set to 1 or more, it will decimate, or only write on multiples of that number. For example, a decimation rate of 2 would cause the block to be read or written once for each two iterations of the PCL. A PCL rate of 100 Hz would then have the block read or written at a rate of 50 Hz.

# Adding and configuring a Channel

When the Add Channel button is pressed, a new channel is created in that block. Remember that the read or write direction is inherited from the block – it cannot be set on a per-channel basis.

Channels can be configured on their page:



* **Name:** You can give each channel a name
* **Relaltive offset in block:** This corresponds to the memory address, but it is simplified. It is only relative to the block it is in. Thus, the first channel in a block can be 0, the second channel can be 1, and so on. The same thing is true about these memory address sizes as with block channels: namely, they are in doubles. To find the absolute memory address of a channel, you would multiply this number by 8, and then add it to the block address (which is the block number multiplied by 8).
* **Type:** read or write. This is not settable at this level – only at the block.
* **Default value:** this is the value this channel will have at the start of execution
* **Unit:** this is useful for bookkeeping, but will not directly cause any conversion operations to happen.
* **Block address:** This is the address of the block (see above section for more)
* **Description:** you can enter a description here

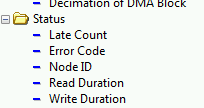
**Note on sequential channels:**

As I mentioned in the start, a DMA operation is a buffered exchange of a contiguous set of memory addresses. That means that all of the channels are exchanged, and one block corresponds to one exchange.

The note on this page modifies this slightly – or really, adds more detail. Channels to write must be in a row. Channels to read can have space in between. So for writing channels, nothing is new. But when reading channels, we can skip? What’s going on? Simply that the intervening addresses are also read. For example, if we read a channel at 0 and one at 10,000 then all 10,000 channels will be read. This is not a problem for small gaps, but keep this in mind for large datasets. The data is still being exchanged, even though it might not all be finding its way into VeriStand channels.

# Status channels

The custom device has several status channels. These are updated at runtime to inform you of the status of the code:



* **Late count:** This is the count of the number of times this custom device loop finished late. If this is increasing steadily, it is not keeping pace with the rest of the VeriStand system. In that case, re-evaluate the configuration, and how things are meeting requirements.
* **Error code:** If the custom device experiences an unhandled error, it will report it here.
* **Node ID:** This is the Node ID of the GE reflective memory card. This value is set on DIP switches on the card itself. Note that if there are two physical channels with the same Node ID on the same bus, they will not be happy. I believe Node ID 0 will also cause the cards to be grumpy.
* **Read Duration:** This is the time in microseconds that it took to perform the last read
* **Write Duration:** The time in microseconds it took to perform the last write

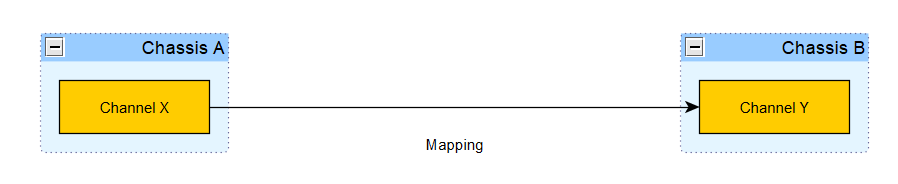
# Coming from VeriStand

VeriStand’s built-in Data Sharing feature for reflective memory hides most of the details that this custom device exposes. Addresses, decimation rates, and more will all be new. There is one additional feature that merits discussion, however: Mapping.

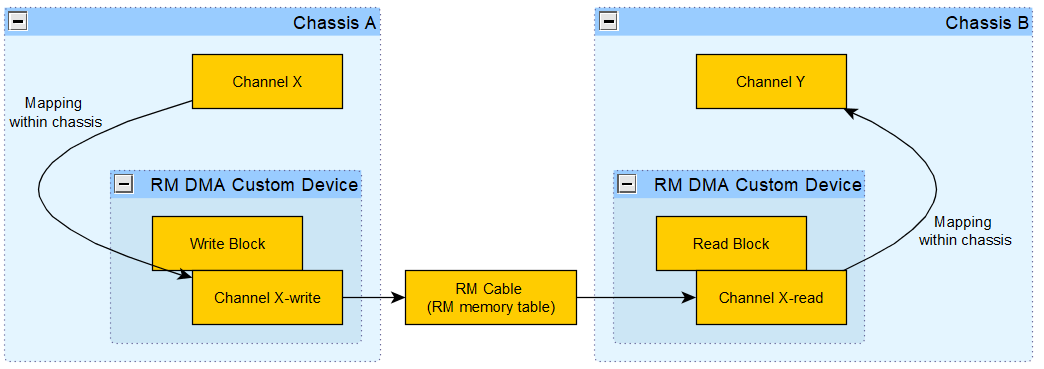
VeriStand’s built in feature makes mapping channels between two VeriStand targets very easy. It abstracts the mapping of channels into and out of the cards, as well as the managing of the memory table. It makes it easy to map a channel(say, channel X) from one chassis (say, Chassis A) to another (say, Chassis B) by directly mapping – as if they were on the same system.

What it is doing behind the scenes involves creating an input on the first custom device, mapping the channel over to it, creating a memory map, giving it to both cards, creating an output on the second custom device, and finally mapping it to the destination channel.

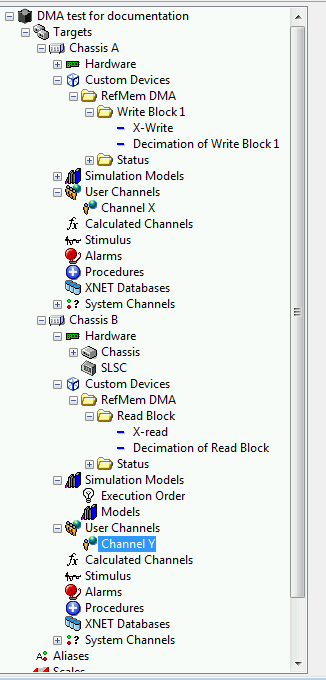
Here is conceptually what happens with the built-in feature:



And here is the same end goal, accomplished with this custom device:



Here is how it may look in the system explorer (using User channels as placeholders for Channel X and Y:



It’s relatively straightforward once you realize that you have to map the channel into the reflective memory card and back out again on the other side.

Just remember to make sure the configurations mirror each other. (Again, that’s part of why CSV files are helpful.)