GBTx Communication Document

University of Maryland LHCb group

April 19, 2018

`Rev: 0.3.1 (63deb45)`

Contents

1 Hardware setup **2**

1.1 Overview 2

1.2 Configure GBTx to use external I²C adapter 2

1.3 Configure slave GBTx to use master SCA channel 2

1.4 Configure GBTx to use GBT-IC channel 3

1.5 Reset GBTx 4

2 Software setup **4**

2.1 Program master GBTx with external I²C adapter 4

2.2 Check communication between master GBTx and MiniDAQ . 5

2.3 Program individual registers of slave GBTx 6

2.4 Program slave GBTx with configuration files 7

Appendices **8**

A Fix “Waiting for a GBT server to run” 8

B Bypass components in GBT data path 8

1 Hardware setup

1.1 Overview

Our current setup consists of one master and one slave GBTx board. The master is connected to the MiniDAQ GBTx channel 3 (fiber 8), and the slave can be connected to either GBT channel 0 (fiber 6) or channel 6 (fiber 11). The master synchronizes its on-board clock to the signal from the MiniDAQ, and propagates its clock signal to the slave. The slave does not have an on-board clock, and is configured to obtain clock signal externally.

The master I²C port is connected to an external USB device. The slave I²C port is connected to the master. Both are set to be programmed by the I²C channel, rather than GBT-IC channel.

The current setup is capable of:

1. Program the slave GBTx board with MiniDAQ directly.
2. Read/Write the register value of the master GBTx board with GBT-IC specification on the MiniDAQ.
3. Do PRBS tests from MiniDAQ to the slave, then back to the MiniDAQ. The master is also required[1] as the slave can only obtain its reference clock from the master.

1.2 Configure GBTx to use external I²C adapter

This setup is required to program a GBTx board using an external I²C adapter. Follow Figure 1 to connect an external I²C adapter.

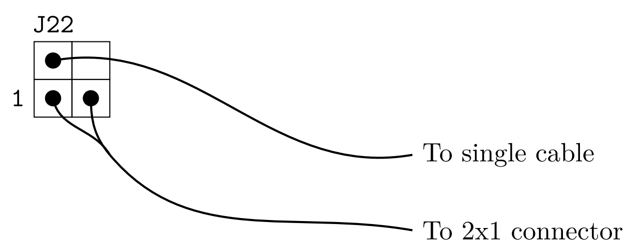

Figure 1: Schematic for external I²C adapter setup.

> The I²C adapter used in our lab is made in-house. For the 2x1 connector, make sure the side that has *no* metal contact is facing up.

1.3 Configure slave GBTx to use master SCA channel

This setup is required to program a slave via a master SCA channel. In a typical scenario, the master is connected to a MiniDAQ so that programming the slave using the MiniDAQ directly[2] is possible. Follow Figure 2 to connect a slave GBTx to the SCA channel of a master GBTx board.

[1] The master GBTx is also connected to the MiniDAQ with a different channel, to provide reference clock to the slave.

[2] MiniDAQ → master GBTx → slave GBTx.

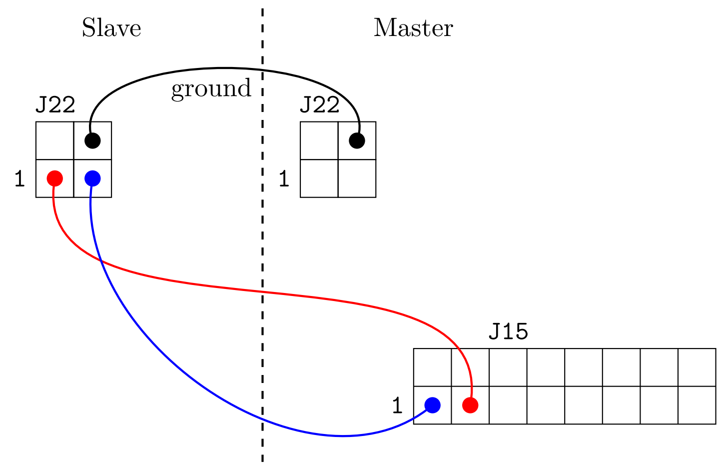

Figure 2: Schematic for slave to master SCA setup.

The black ground cable can be connected to any of the ground pin on the master GBTx.

There is a 2x1 to 2x1 cross-type cable made in-house to replace the red-blue cables. To use that cable, make sure the two 2x1 connectors have the same orientation (e.g. the sides *without* metal contact are both facing up).

1.4 Configure GBTx to use GBT-IC channel

It might be useful to read/write individual registers from/to a GBTx board. In this case, follow the Figure 3 to flip the `configSelect` switch.

Flip the `configSelect` switch will render the external I²C adapter ineffective. None of the GBTx register value is fused onto the board, so a GBTx board in our lab must always be programmed externally via I²C before flipping the switch.

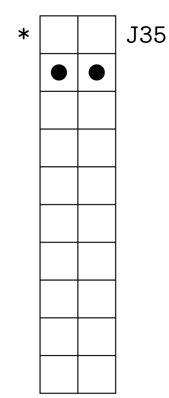

Figure 3: Schematic for flipping the `configSelect` switch. A jumper should be used to connect the two pins marked above.

1.5 Reset GBTx

Sometimes GBTx boards will not be properly reset by reprogramming. In such case, a hard reset is needed. Follow the Figure 4 to reset GBTx boards.

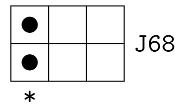

Figure 4: Schematic for resetting GBTx boards. A jumper should be used to connect the two pins marked above.

2 Software setup

2.1 Program master GBTx with external I²C adapter

Before proceed, follow the instruction on subsection 1.2 to configure the hardware first. As said in the previous section, the master GBTx board must be programmed via an external I²C adapter first. A Windows 7 computer on the rack is used. The **GBTX programmer** is located at:

```
DT_Rack\GBTx_programmer\GBTxProgrammer.jar
```

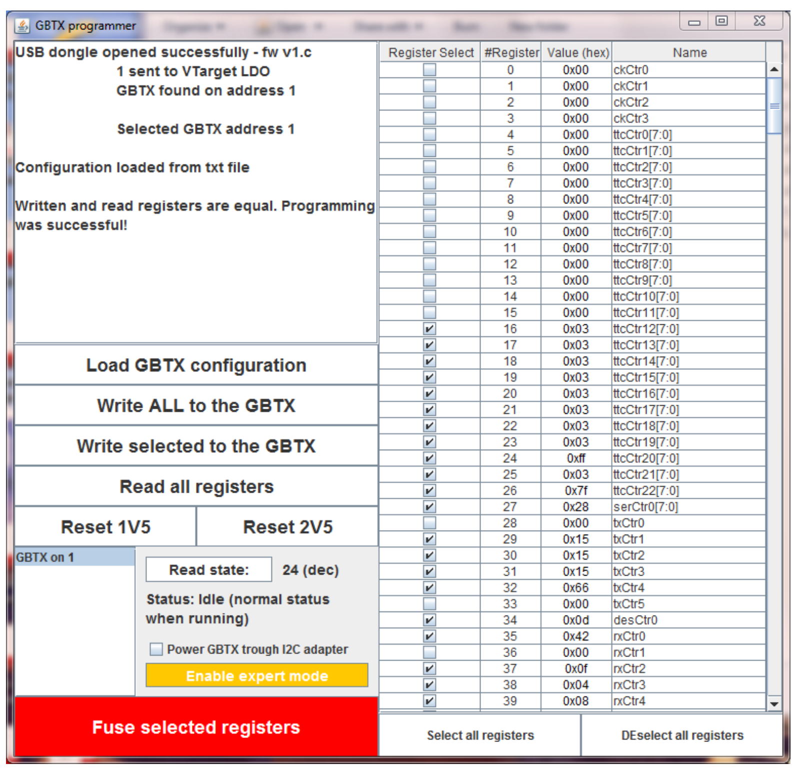

Figure 5: A typical UI for GBTX programmer.

Launch the programmer, a typical UI is shown in Figure 5, Click **Load GBTX configuration** and load a configuration file, which is located at:

```
DT_Rack\GBTx_programmer\GBTx_TRx_v12_test_withWatchDog.txt
```

Then click **Write ALL to the GBTX**. Check the returned message to make sure everything works (supposedly). Now click **Read state**. If the master GBTx is configured correctly and is connected to a working MiniDAQ, the return value should be:

```
24 (dec): Idle (normal status when running)
```

2.2 Check communication between master GBTx and MiniDAQ

After program the master GBTx board and verify the return value, we can check the communication between the GBTx and the MiniDAQ with **GBT Client**.

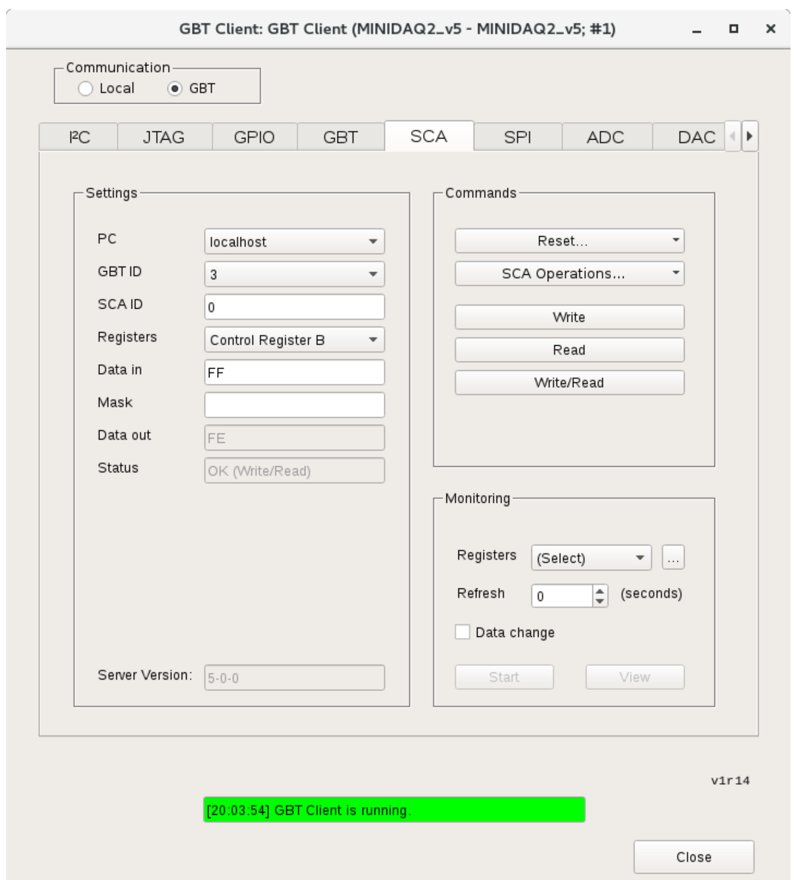

Figure 6: A typical UI for GBT Client, SCA tab.

Here we need to use the Linux server on the rack. To launch that program, locate the **gedi** panel. Under **JCOP framework**, click the **GBT Client**. Choose **GBT** option under the **Communication** tab. Navigate to **SCA** tab.

To check whether the link between GBTx and MiniDAQ is successfully established, configure the parameters *exactly* as shown in Figure 6. Now click **Write/Read**. The **Data out** field should have a value of "`FE`", and the **Status** should be "OK (Write/Read)".

> **GBT ID** corresponds to the physical optical link that is connected to the master GBTx board. Recall that in our setup, the master is connected to optical fiber 8, which is mapped to GBT channel 3.

> There are only 4 **Registers**. All 4 registers work, the "Control Register B" is chosen for no apparent reason. What matters is the return value should be consistent with our expectation.

2.3 Program individual registers of slave GBTx

Before proceed, follow subsection 1.3 to set up hardware. Again we use **GBT Client** to program individual registers of the slave GBTx. Configure the parameter *exactly* as shown in Figure 7.

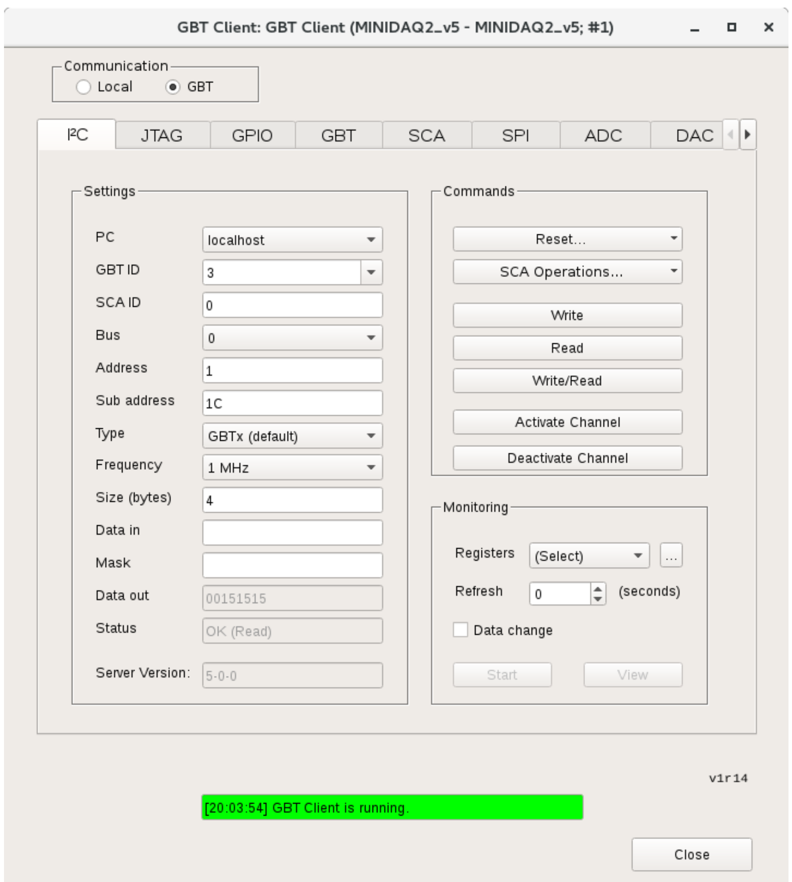

Figure 7: Correct parameters to configure GBT I^{2}C tab.

> If the **Write** command failed, try **Activate Channel** first.

> **Address** corresponds to GBTx index, which is configured via a onboard DIP switch. Default to 1.

> Recall that 1 *Byte* is 8 *bits*, and 1 Byte corresponds to a 2-digit hex number.

> Notice in the **Data out** field, the last 3 numbers are the same. This is a requirement in the GBT specification that the configuration registers must be triplicated to minimize error.

2.4 Program slave GBTx with configuration files

The setup is exactly the same as shown in subsection 2.3. All we need to change are replace the **Size** field with "0", and provide the configuration file location with the following syntax, which needs to be filled in the **Data in** field:

```
file:<path_to_configuration_file>
```

> This is equivalent to program the slave with an external I²C adapter. The advantage of doing this way is we don't need to remove the I²C adapter from the master. In other words, once the hardware setup is done, we don't have to touch them anymore; whereas if we program both master and slave with an I²C adapter, it must be moved around between the master and the slave.

Appendices

A Fix "Waiting for a GBT server to run"

Sometimes when we launch the **GBT Client**, it would claim that it is "Waiting for a GBT server to run". Take this warning literally: It means that currently there is no GBT server that is alive. The fix is easy: fire up a terminal, and type in "`GbtServ`".

B Bypass components in GBT data path

As shown in Figure 8, we can bypass components in GBT input/output path.

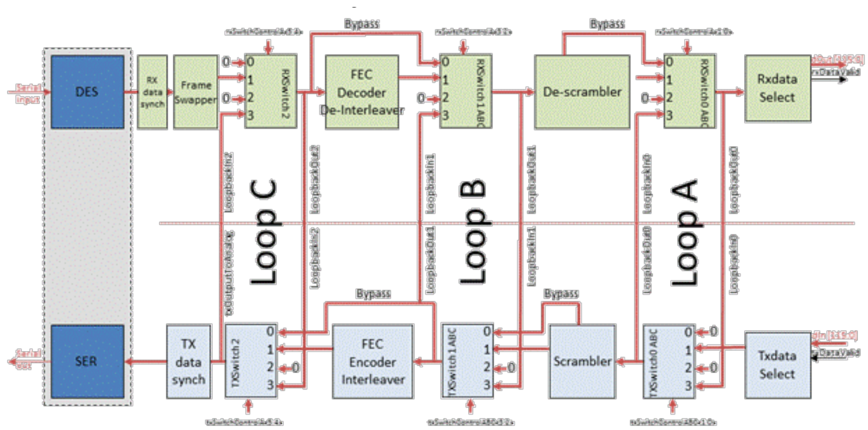

Figure 8: GBT data path. Stolen from LHCb *GBTX manual*.

The register `0x1c` and 3 subsequent registers control the output path; `0x2d` controls the input path. Each of the paths have 3 configurable selectors, each has 4 operations modes ($0_{10} - 3_{10}$[3], which translate to $0_2 - 11_2$). According to Figure 8, the control sequence should be given in:

```
TXSwitch2 -> TXSwitch1 -> TXSwitch0
```

Here is an example. Let us consider the Tx normal operating case (i.e. all 3 switches are configured to 1). We have:

$$\underbrace{1_{10}}_{\text{TXSwitch2}} \; \underbrace{1_{10}}_{\text{TXSwitch1}} \; \underbrace{1_{10}}_{\text{TXSwitch0}}$$
$$\Downarrow$$
$$\underbrace{01_{2}}_{\text{TXSwitch2}} \; \underbrace{01_{2}}_{\text{TXSwitch1}} \; \underbrace{01_{2}}_{\text{TXSwitch0}}$$

To convert binary 010101_2 to hexadecimal, we left pad the result with 2 additional 0's: $0001, 0101_2$. The end result is 15_{16}.

[3] The subscript indicates the base.

15_{16} is exactly the last 3 bytes in **Data out** field in Figure 7. This is not a coincidence.

Recall that each 4 digits in binary exactly correspond to 1 digit in hexadecimal.