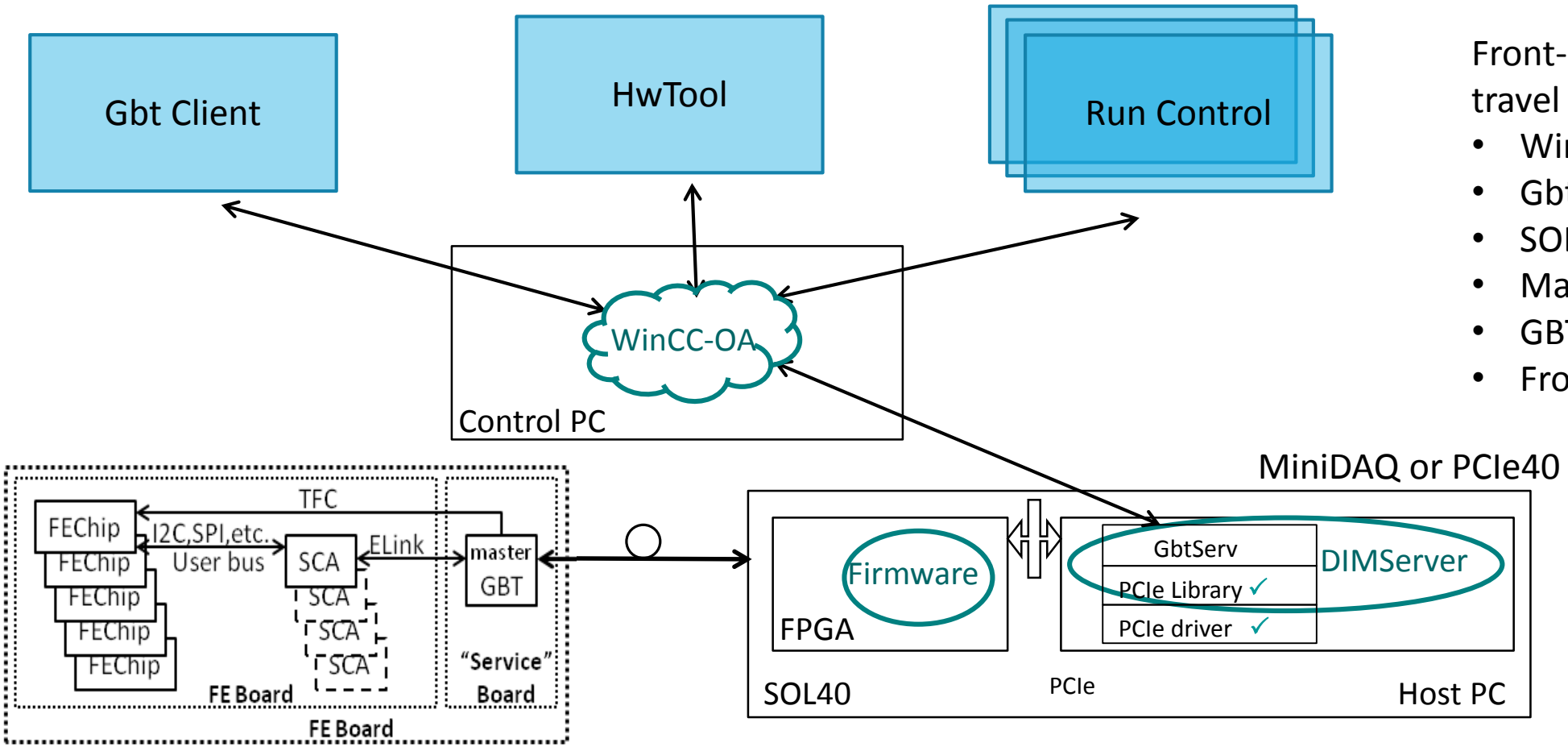
FE Board configuration

JOÃO BARBOSA

MINIDAQ2 WORKSHOP

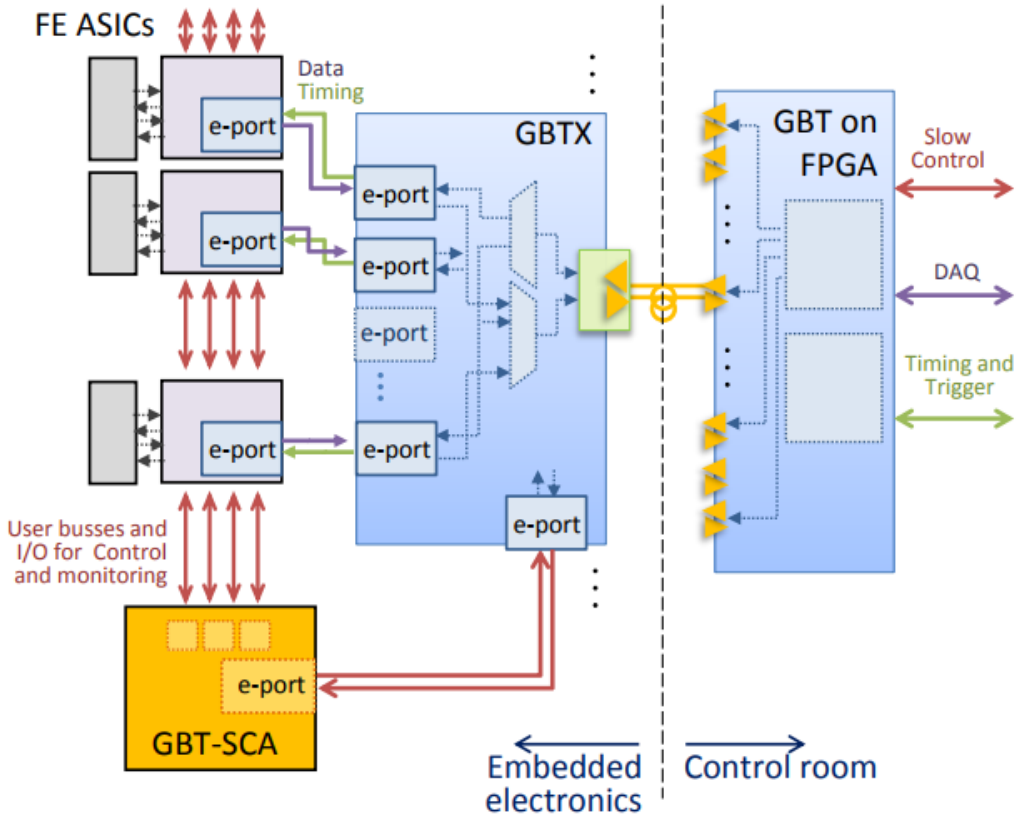
Control the GBT-SCA (and few other things)



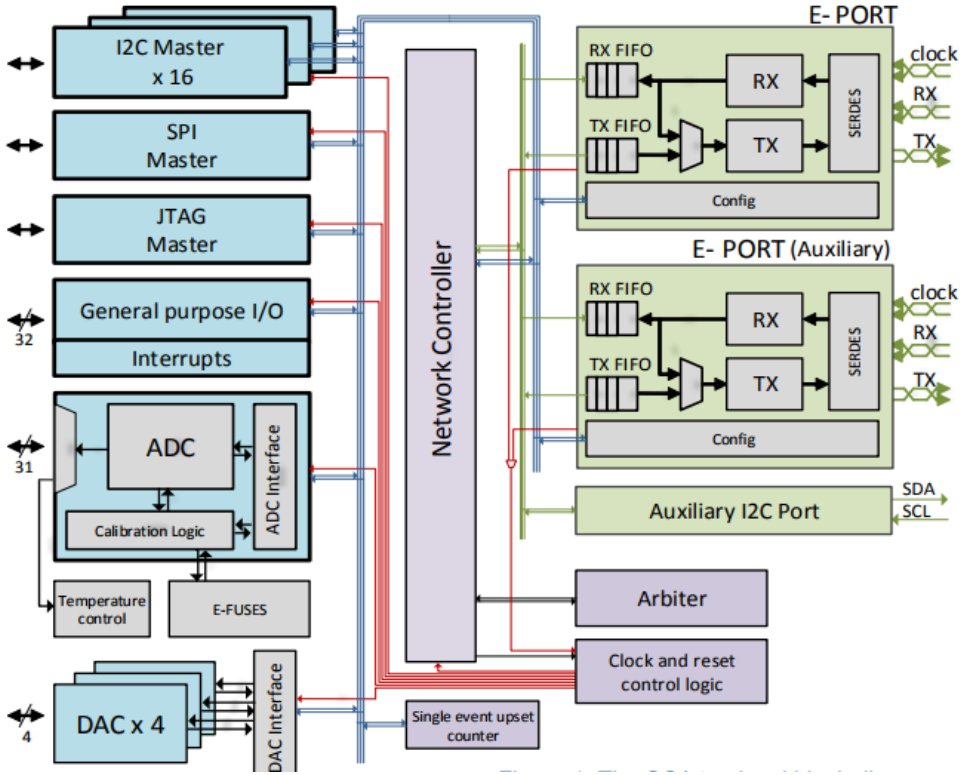
Front-End slow-control commands travel through:

- WinCC-OA
- GbtServer (DIM)
- SOL40 (specifically SOL40_SCA)
- Master GBT
- GBT-SCA
- Front-End

The GBT-SCA



GBT-SCA in the GBT system



GBT-SCA schematic

The SOL40_SCA core

- Firmware core inside the SOL40
- Receives commands from the GbtServer through registers in BAR0 (and tools)
- Transmits these commands through the GBT frame into the elink where a certain GBT-SCA is connected

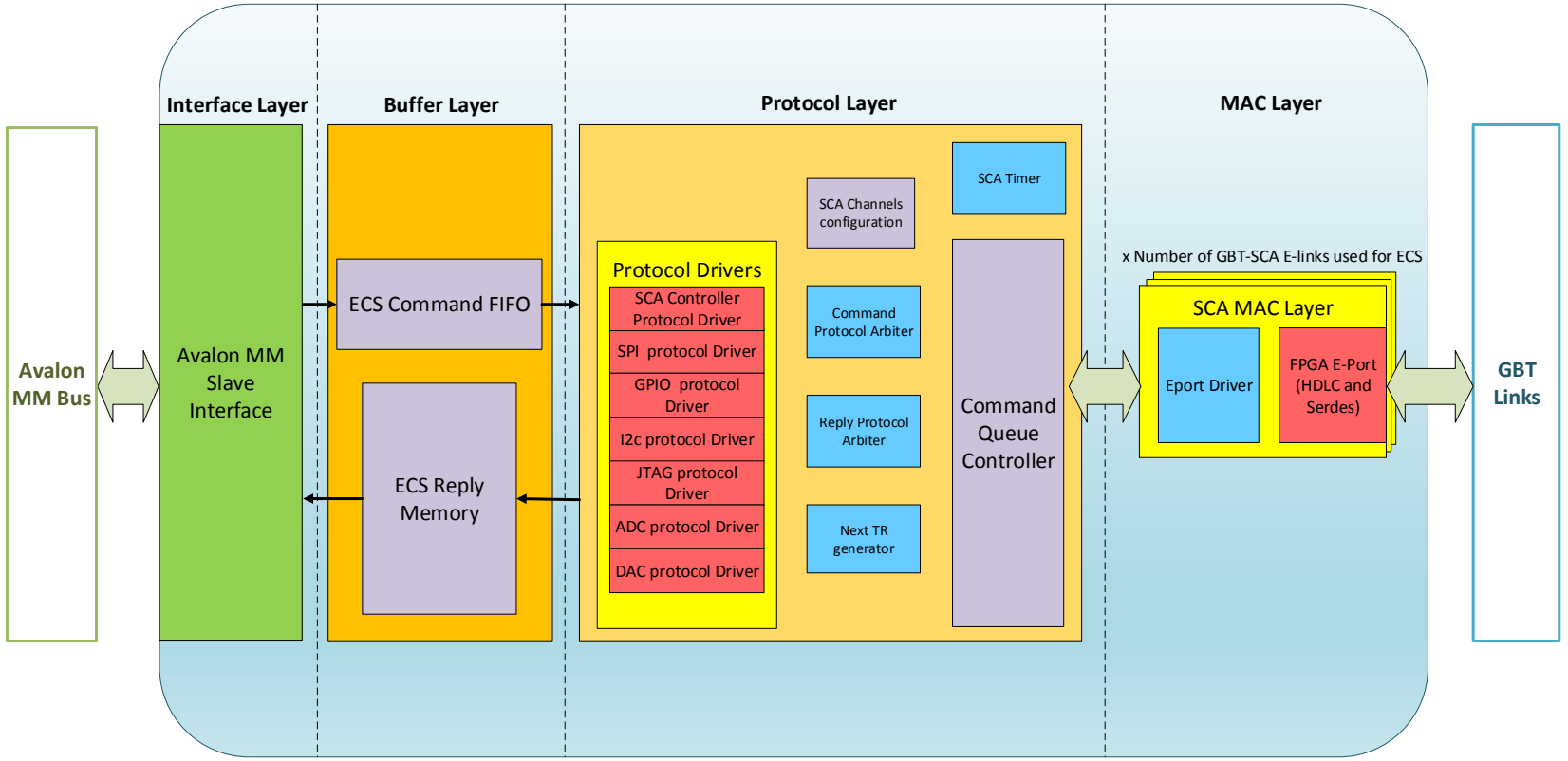
- These need to be instantiated in firmware in a tfc link
  - Look for “constant SOL40_with_SCA_core” in the detector_constant_declaration.vhd file of amc40firmware (lhcb-amc40firmware/mini_daq/mini_daq/packages/test/detector_constant_declaration.vhd)

```
-------------- Fibers connected to ////// SOL40 \\\\\ ----------------------------------------------
constant fiber_mapping_tfc_or_acq       : std_logic_vector(47 downto 0) := x"000000" & x"000001";
--Bit 0 correspond to the fiber link 1, set 1 to this bit if a fiber is connected for SOL40^M
--Bit 1 correspond to the fiber link 2, set 1 to this bit if a fiber is connected for SOL40^M
--    ^M
```

```
constant SOL40_TO_FE_TFC_CMD_offset   : unsigned(11 downto 0) := x"D48";^M
constant FW_with_FE                   : std_logic     := '1';^M
constant SOL40_with_SCA_core          : std_logic_vector(47 downto 0) := x"000000000001";^M
^M
```

The SOL40_SCA core

- 1 instantiation per link
- All protocols for the SCA are implemented in a generic way.
- SOL40 firmware takes about 16% ALMs on Arria 10 for 6 links



Fibre mapping (tested)

Link	Fibre
0	6
1	2
2	4
3	8
4	10
5	12

GbtServer – Features

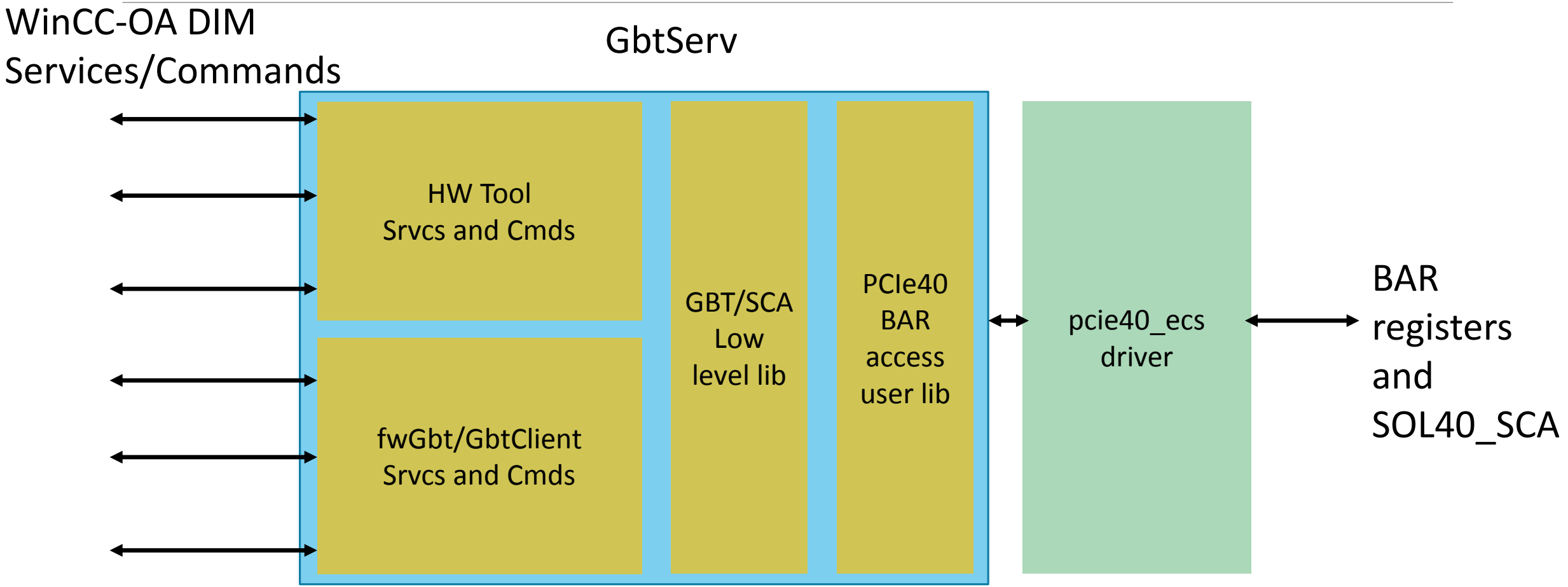
Functionality:

- Read/Write and monitor (polling) hardware registers
- Hw Tool
  - Saves the settings of registers so they can be accessed by name instead of address
- GbtClient
  - Gives direct access to all of the GBT-SCA protocols

Hardware Types:

- Local firmware registers (SOL40, SODIN, TELL40)
- Master Gbt Registers
- GBT-SCA Registers (Activation of FE protocol channels)
- FE Registers
  - I2C (16/sca), SPI, JTAG, GPIO, DAC, ADC

GbtServer - Structure

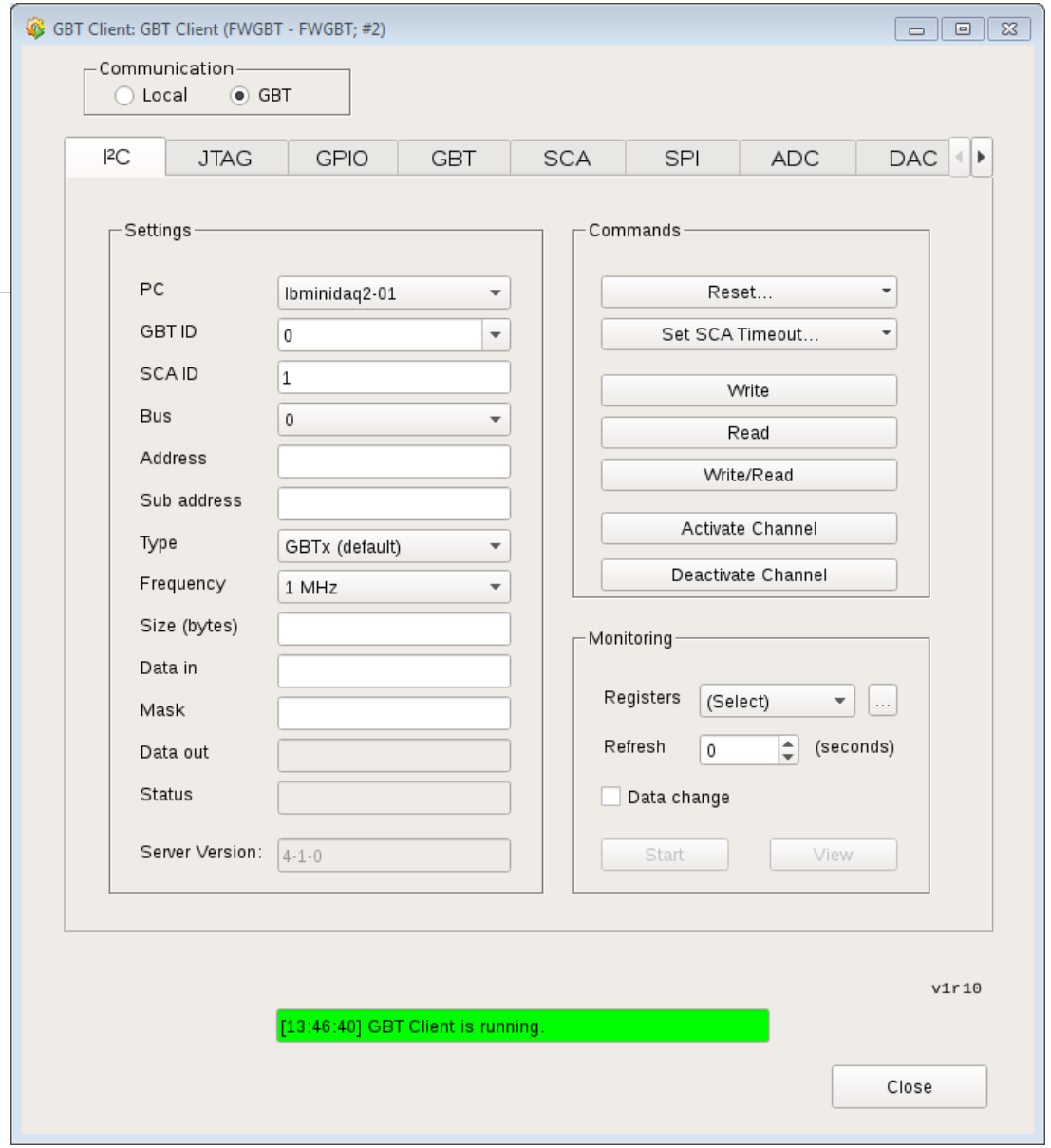


GbtServer – How do I get it

- Install the RPMs contained in either one of these:
  - Lbredmine: https://lbredmine.cern.ch/projects/amc40/files
  - Gitlab repo: https://gitlab.cern.ch/lhcb-amc40firmware-mng/lhcb-amc40software
- You will get
  - GbtServ executable
  - Command line tools meant for debugging the SCA communication (listed in RPM description)
  - Command line tools to configure FPGAs through the SCA

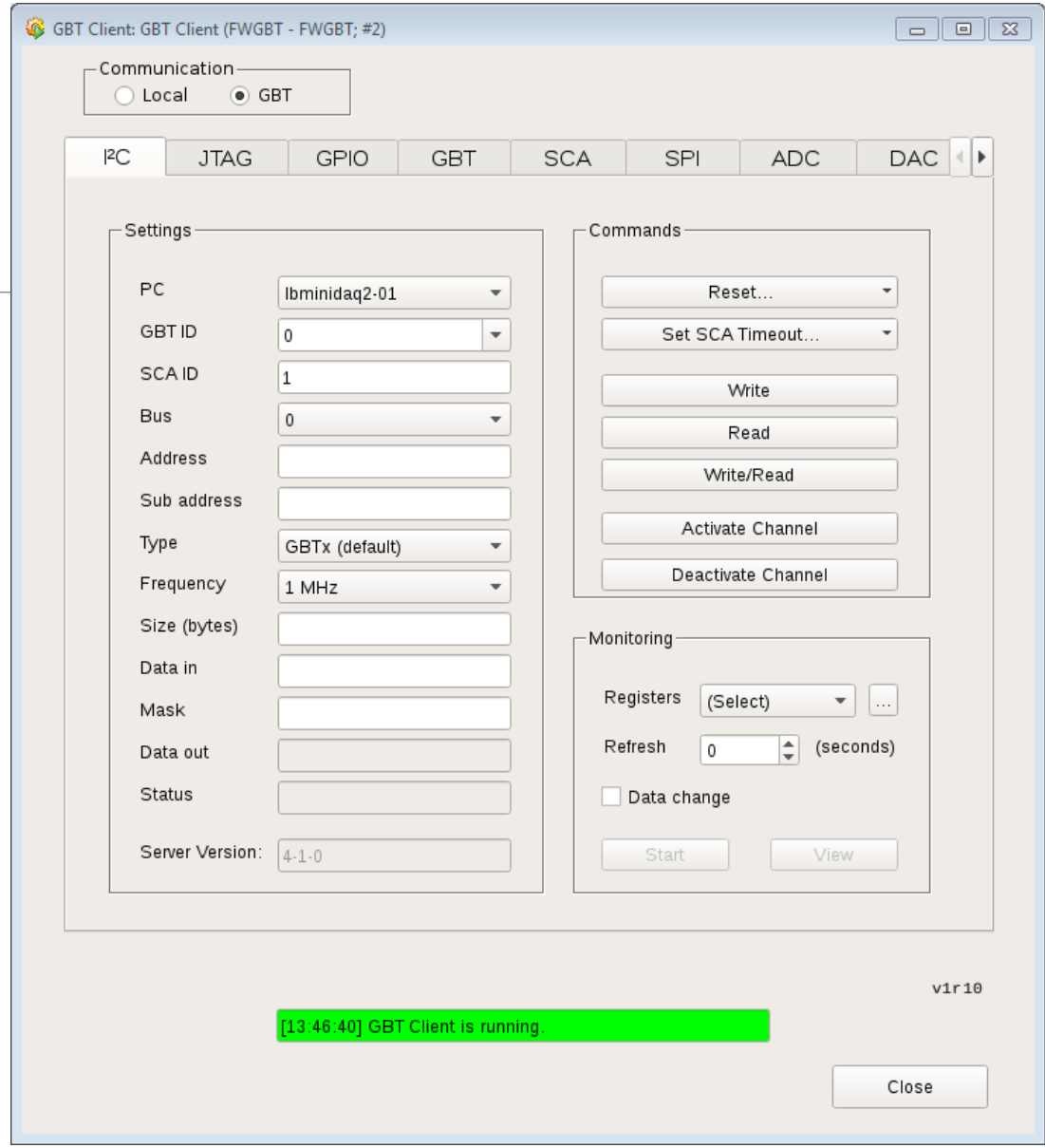
Gbt Client

- Used to test your Front-End register communication mainly
- Uses fwGbt and the GbtServer
- Implements all the GBT-SCA protocols and some more
- Open it in GEDI (“LHCb Framework” -> “Hw” -> “Gbt Client”)



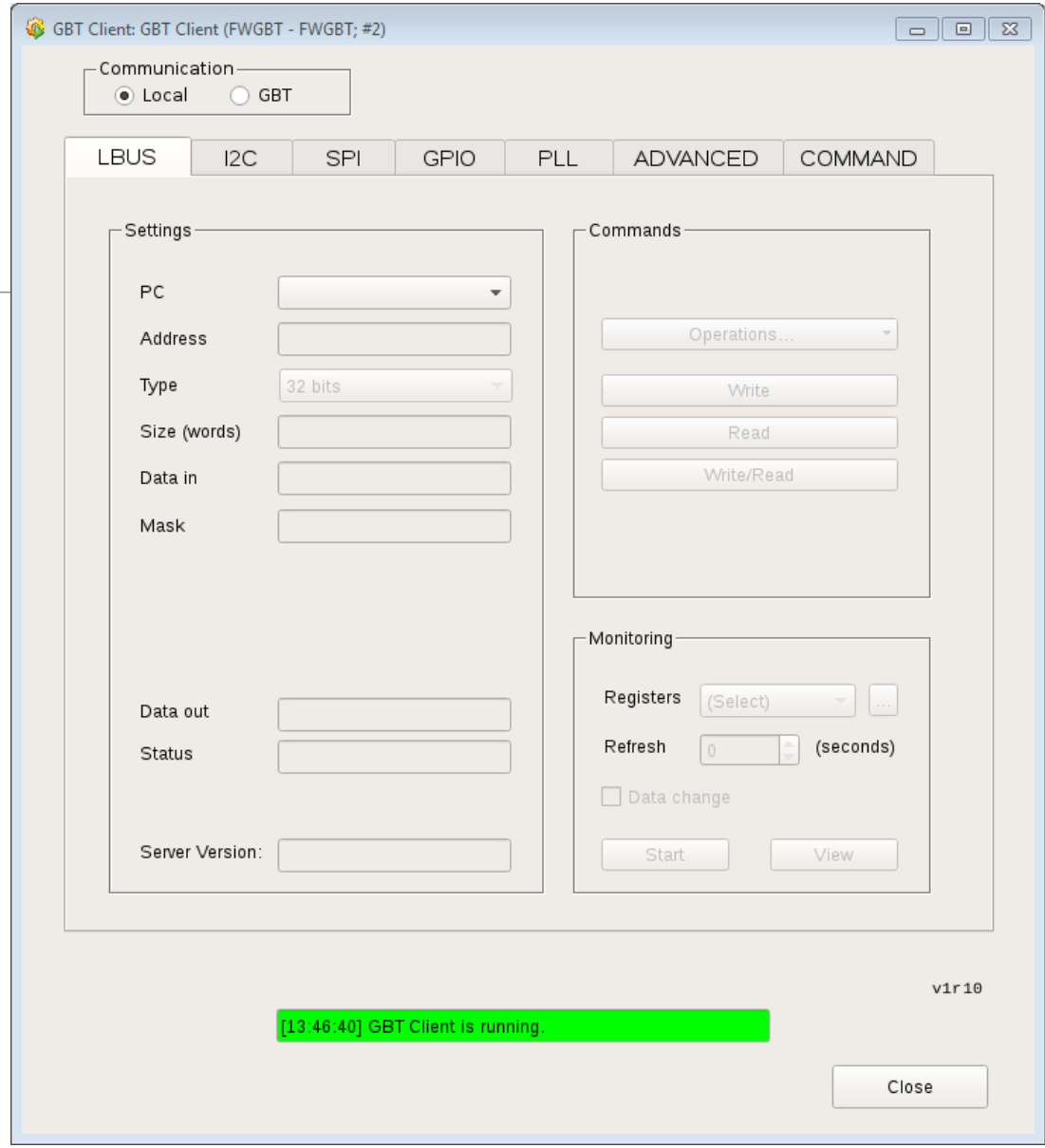
Gbt Client – How to use

- Local and GBT setting



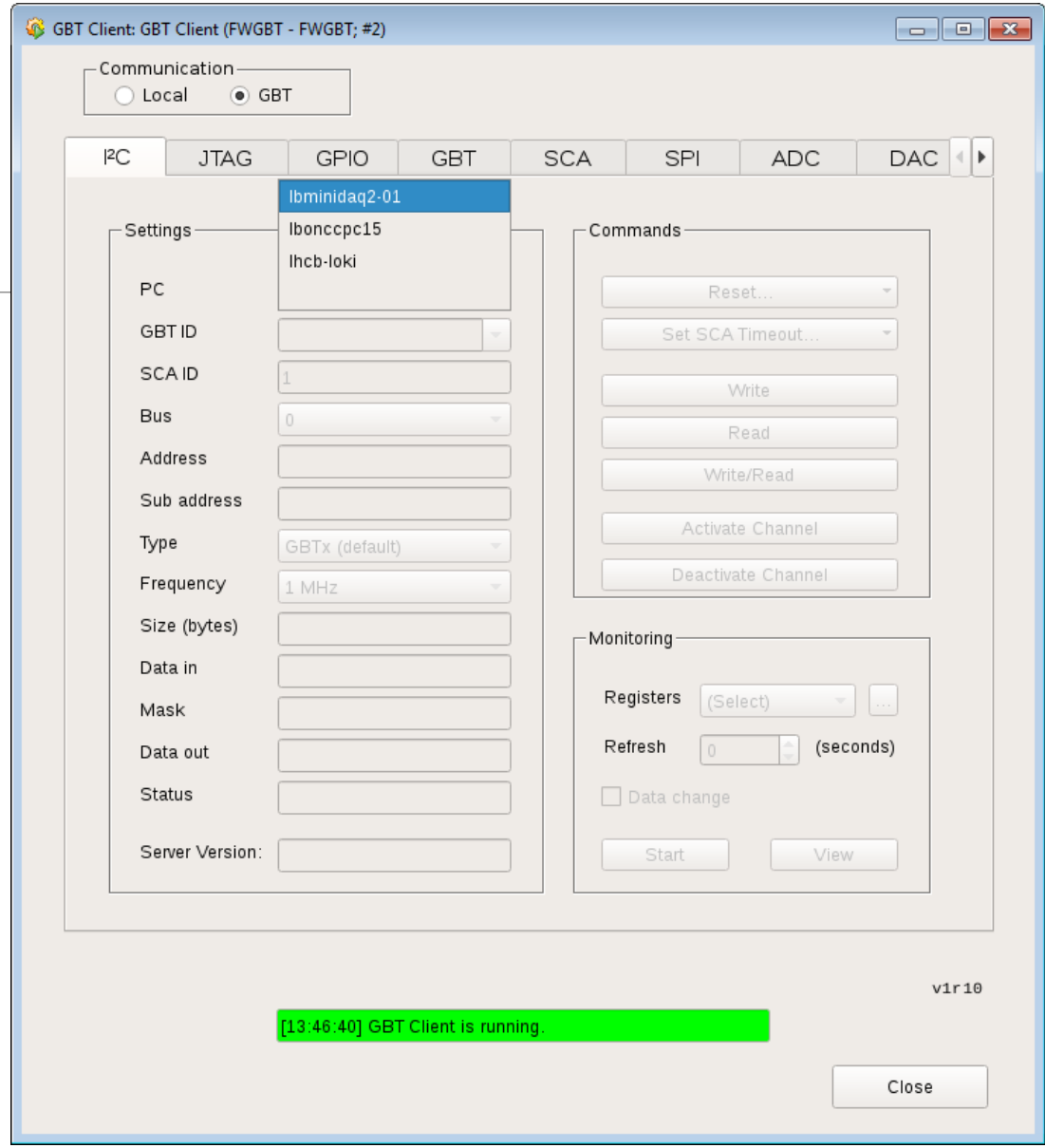
Gbt Client – How to use

- Local and GBT setting



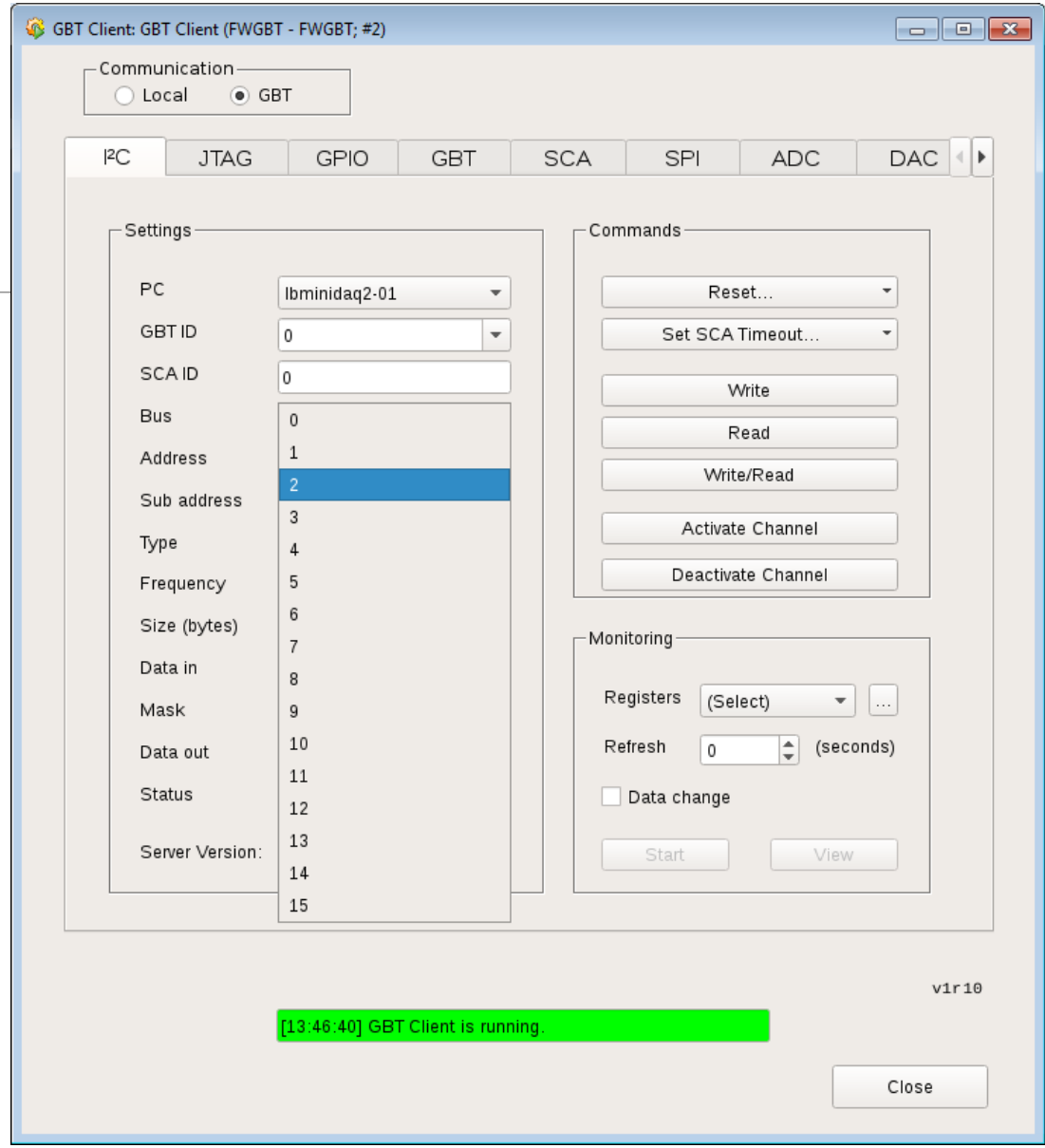
Gbt Client – How to use

- Local and GBT settting
- Choose host of GbtServ



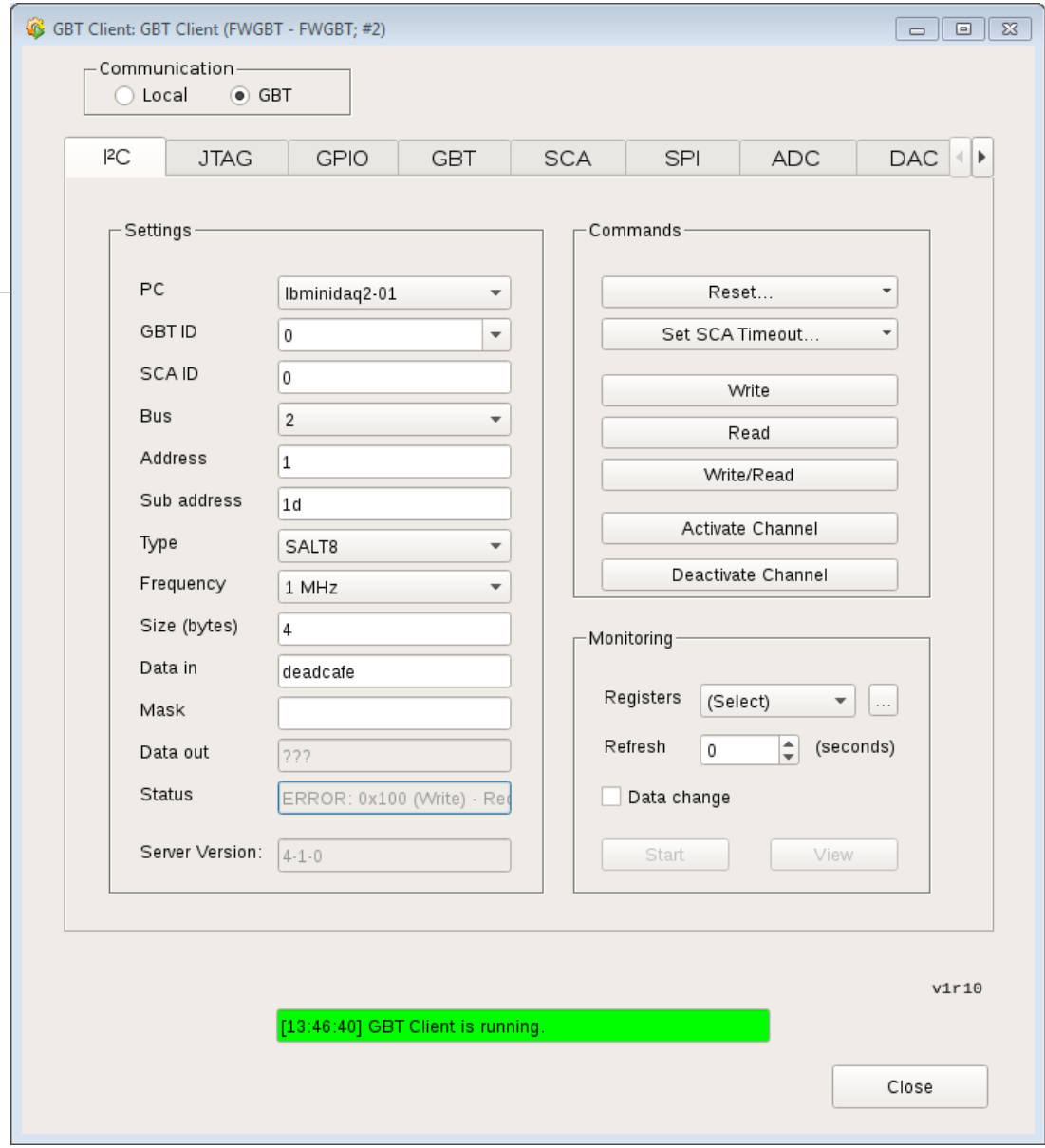
Gbt Client – How to use

- Local and GBT settting
- Choose host of GbtServ
- Set Gbt Link and SCA index (Bus also for I2C) and activate channel if necessary



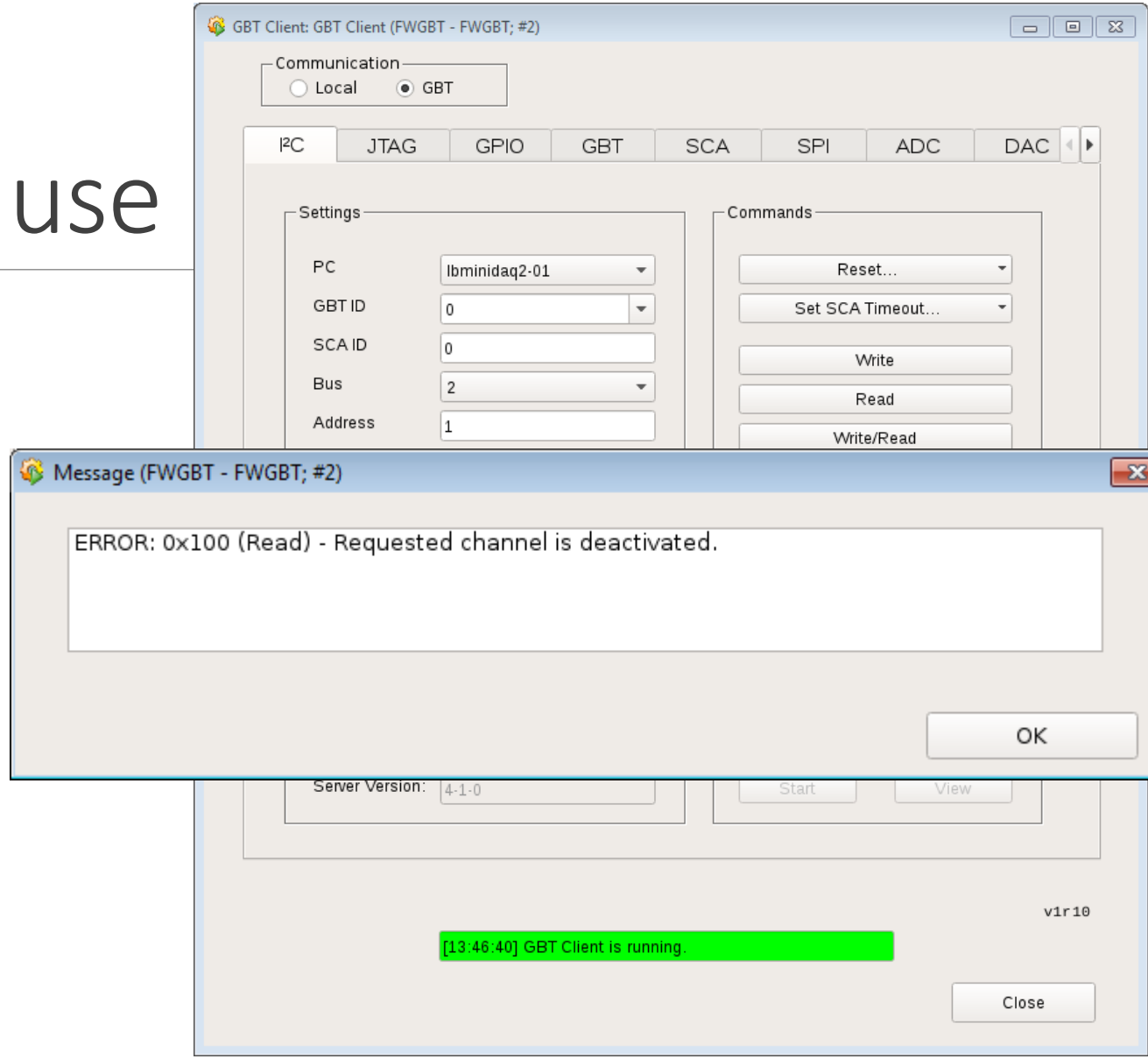
Gbt Client – How to use

- Local and GBT settting
- Choose host of GbtServ
- Set Gbt Link and SCA index (Bus also for I2C) and activate channel if necessary
- Fill in the rest (mask usually uneccessary)
- Perform write/read



Gbt Client – How to use

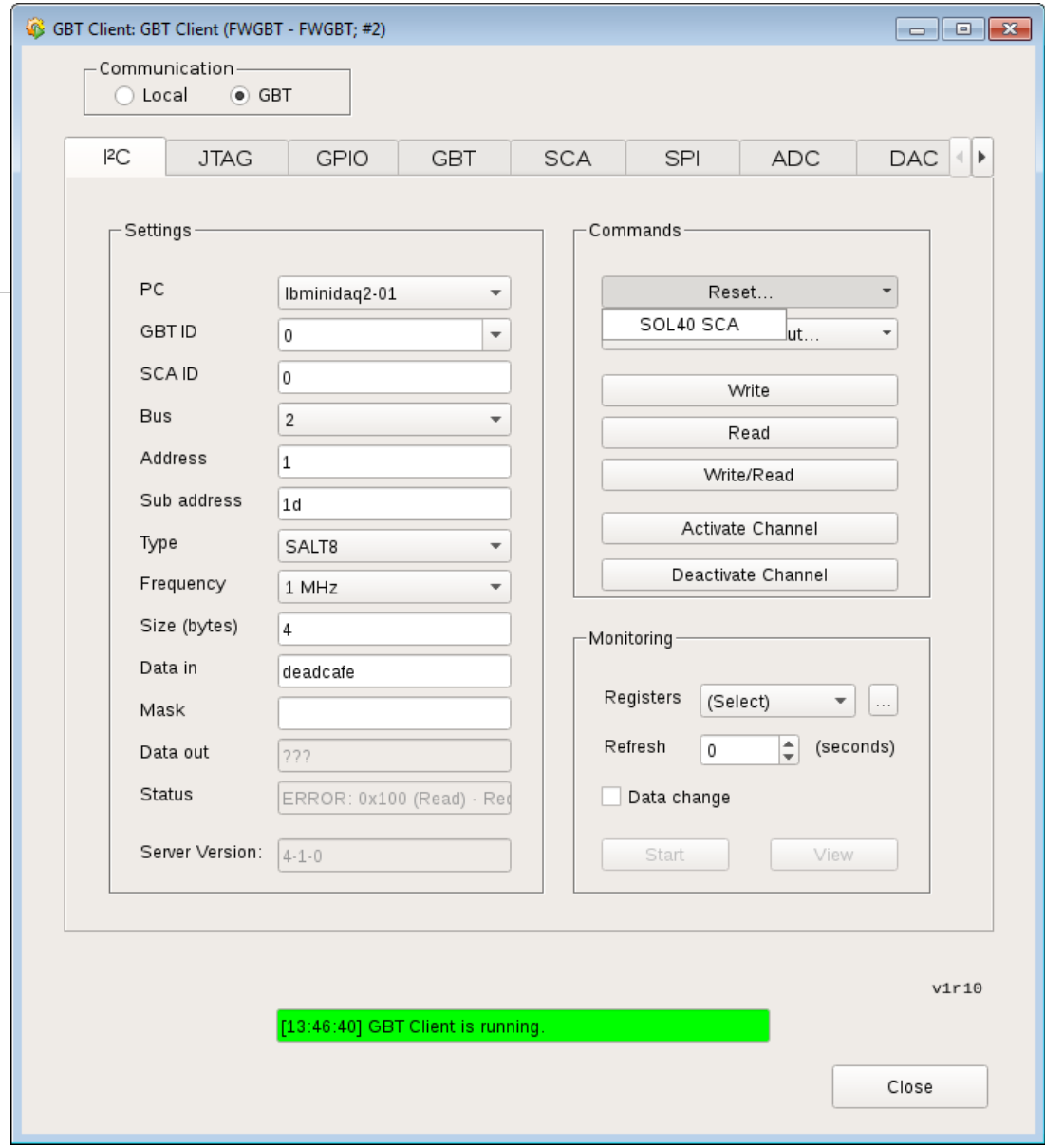
- Local and GBT settting
- Choose host of GbtServ
- Set Gbt Link and SCA index (Bus also for I2C) and activate channel if necessary
- Fill in the rest (mask usually unnecessary)
- Perform write/read
- Check for errors



Gbt Client – How to use

Troubleshooting:

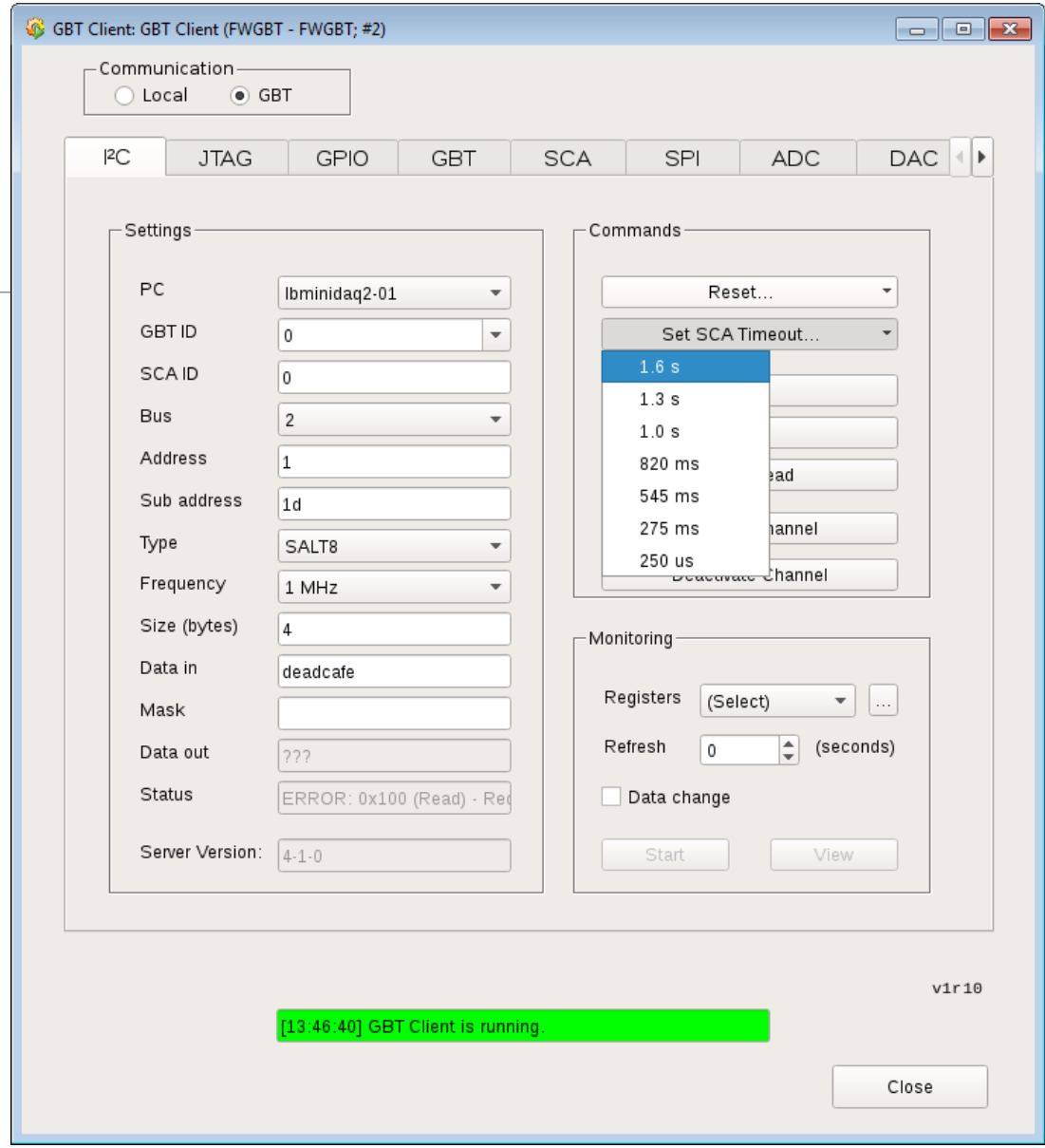
- Reset SOL40_SCA



Gbt Client – How to use

Troubleshooting:

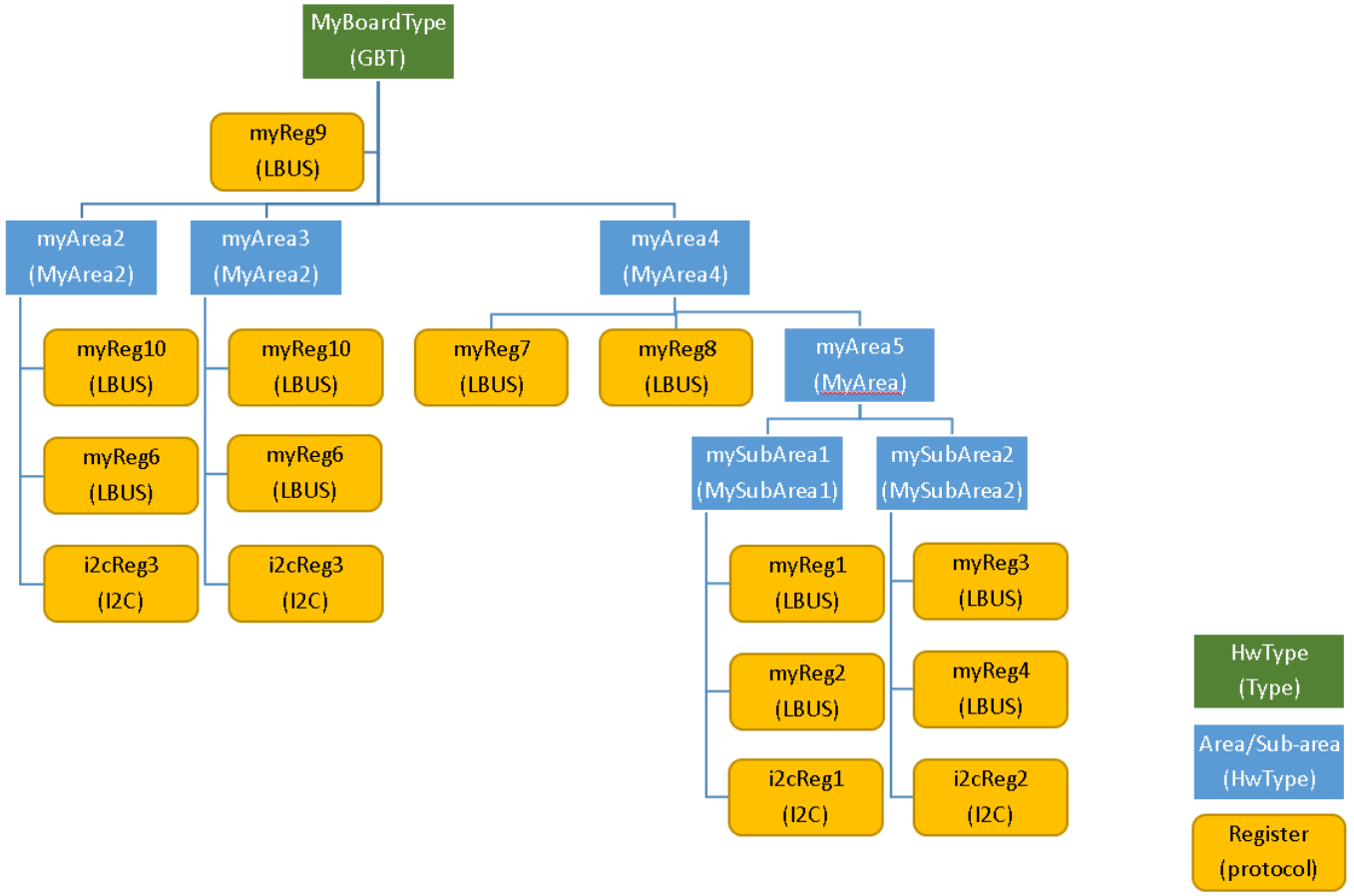
- Reset SOL40_SCA
- Set SOL40_SCA Timeout
- Error descriptions should be very helpful (sometimes)



HwTool - Overview

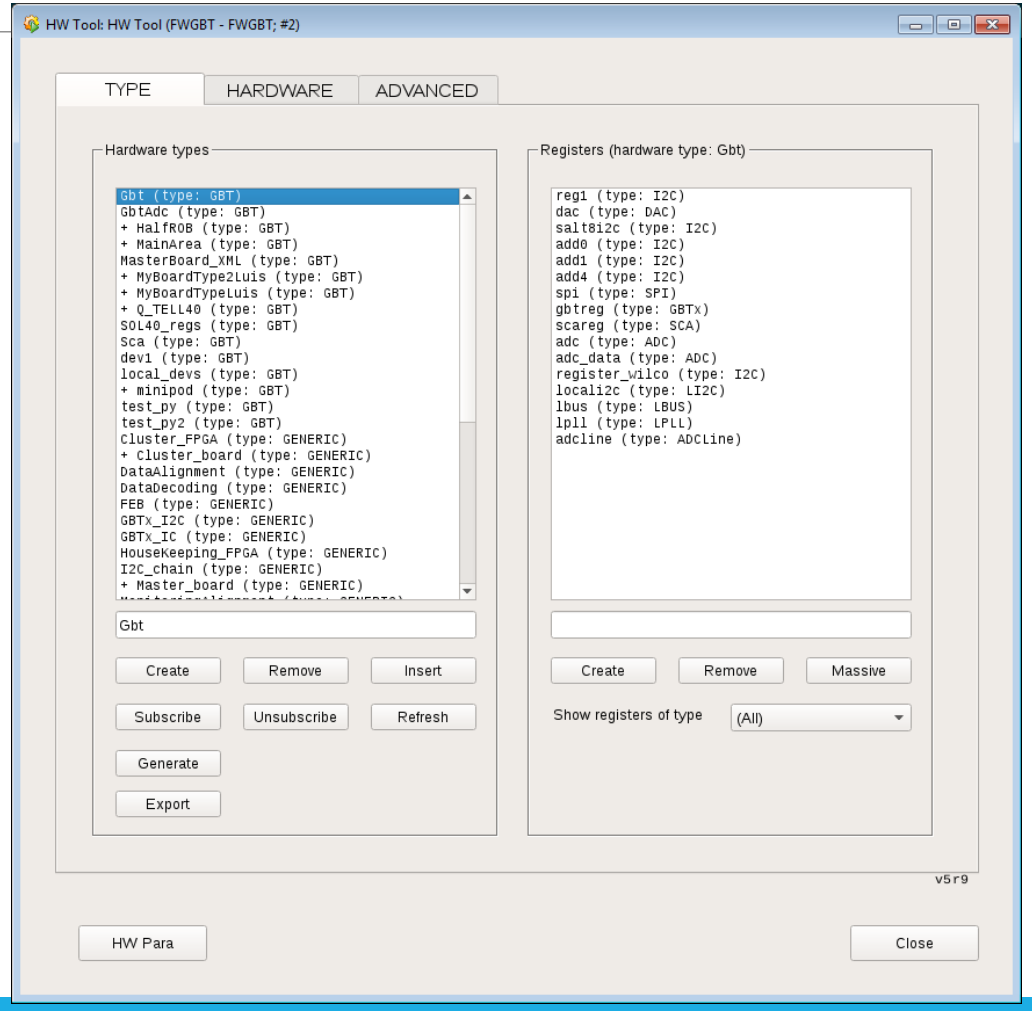
- Allows you to build hardware models (data point structures connected via DIM to GbtServ)
  - Manually through the HwTool GUI
  - **XML file describing your hardware model**
  - Scripting
- Instantiate these types into devices (in the HwTool Panel)
  - You will access these devices making use their instantiation name, meaning…
  - …no need to change your panels even if some board settings change (address, link, sca index, host pc)
- See https://lbredmine.cern.ch/attachments/download/273/Control_System_for_the_MiniDAQ.pdf section 4.3

HwTool – A device model tree



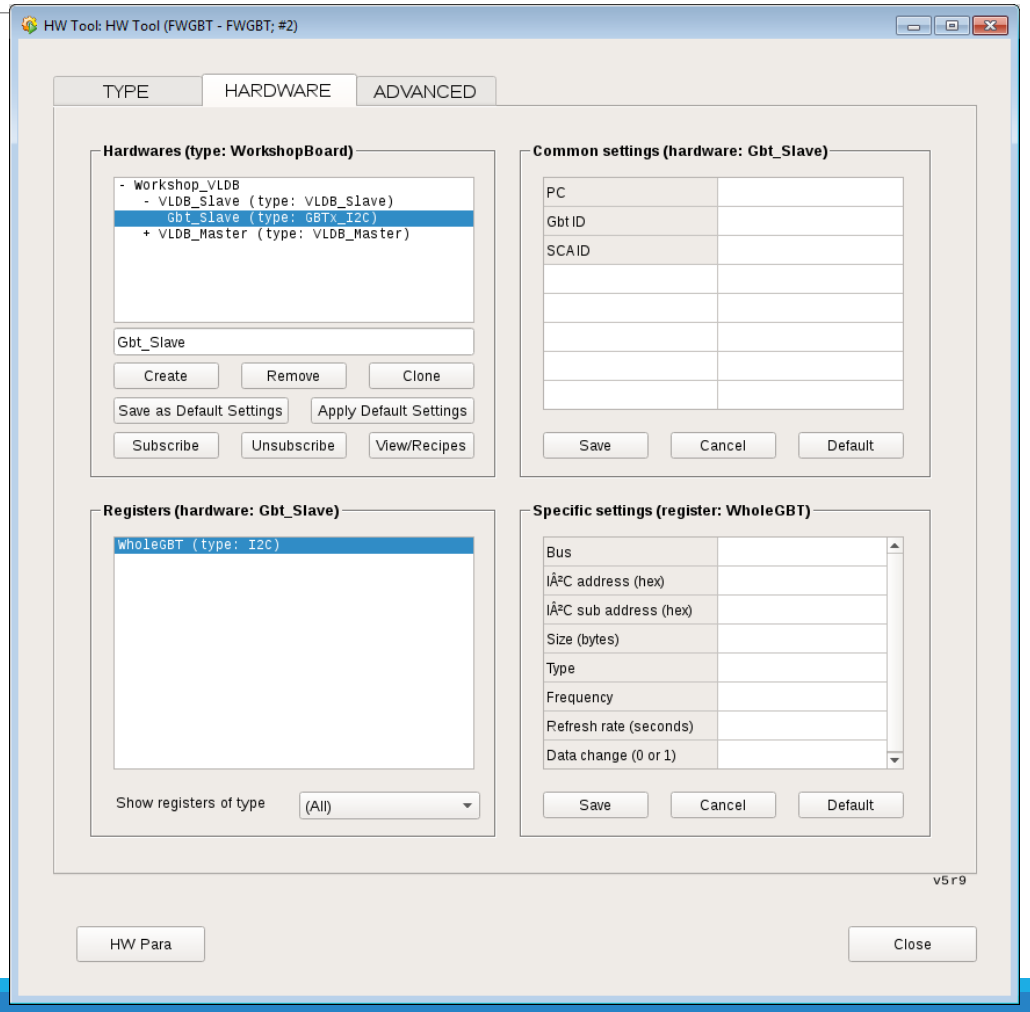
HwTool – GUI (Type tab)

- In GEDI, go “LHCb Framework” -> “Hw” -> “Hw Tool”
- Create device types
- Build a model of a board or group of devices



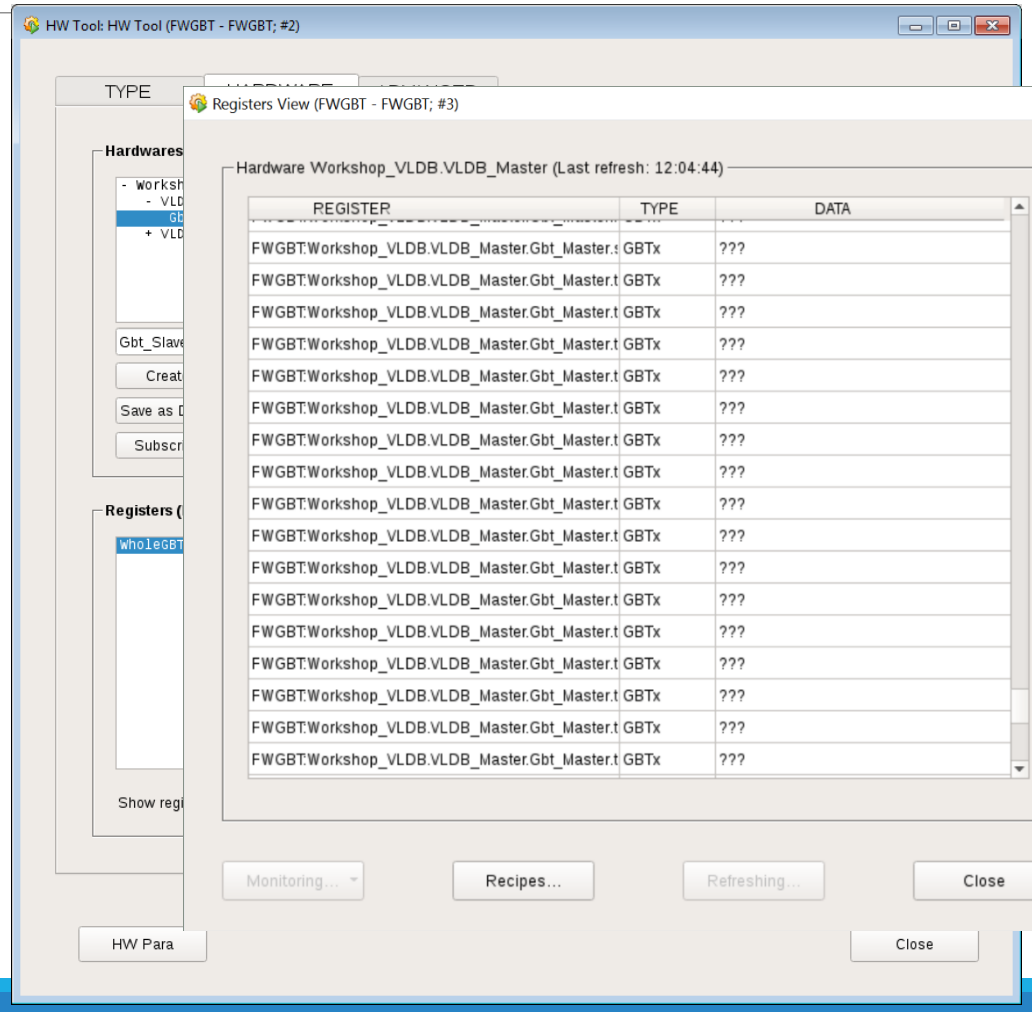
HwTool – GUI (Hardware tab)

- Instantiate devices
- Fill in their settings (painful in this GUI, do an XML!)
- Can apply default settings after you are happy with one instantiation



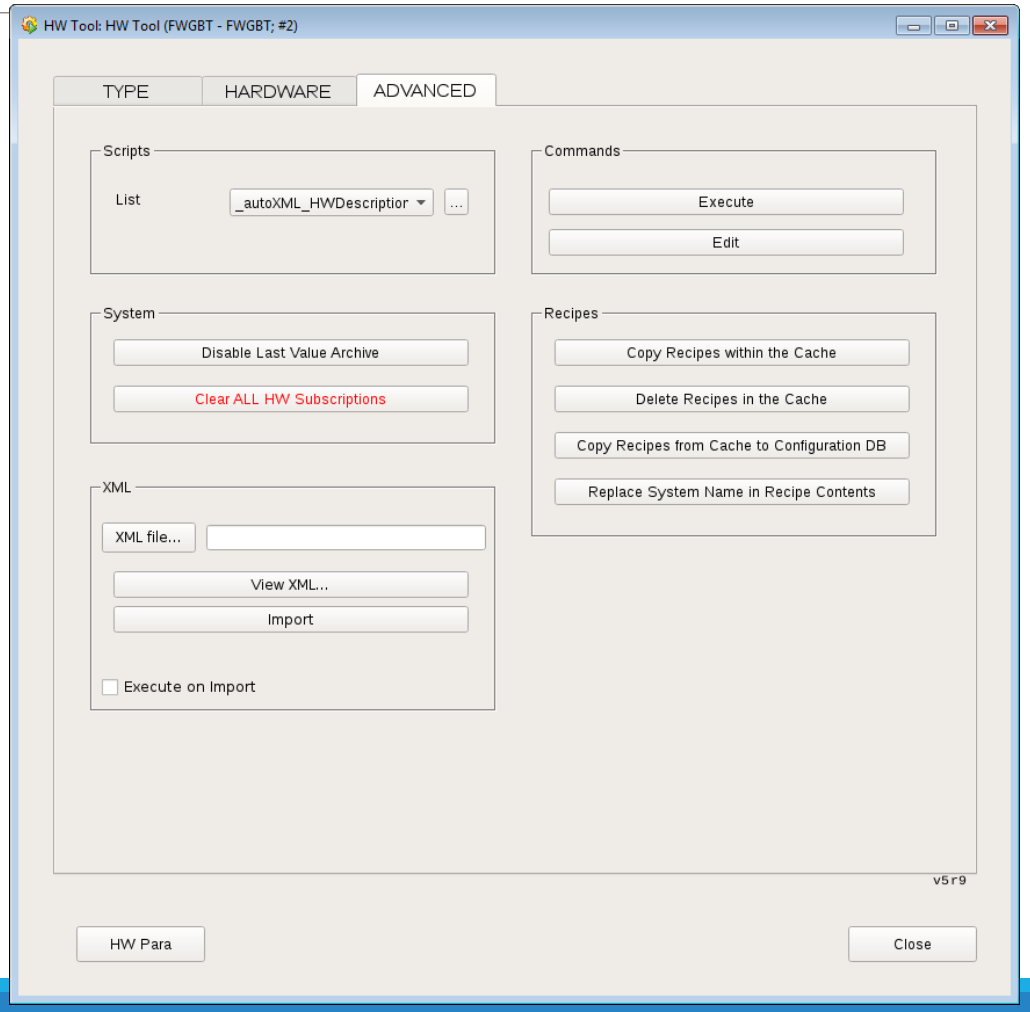
HwTool – GUI (Hardware tab)

- Instantiate devices
- Fill in their settings (painful in this GUI, do an XML!)
- Can apply default settings after you are happy with one instantiation
- Subscribe/Unsubscribe hardware to force GbtServer to instantiate these services
- View/Recipes (Read/Write and start monitoritng a set of registers)



HwTool – GUI (Advanced tab)

- Import XML designs
- Create and Run the instantiation scripts
- Apply Recipes



HwTool – XML files

- Cleaner and more agile way of creating your device models
- Three main elements to consider:
  - hwtype - declaration of an area type and its nodes and/or reg
  - node – declaration of an area or a sub-area inside of an area
  - reg – declaration of a register and its properties
- Check section 4.3.1.3 for details on the various settings

```
<hwtype name="MyArea" type="GBT">
<node name="mySubArea1" type="MySubArea1" address="0x1000000">
<reg name="i2cReg1" type="I2C" bus="2" address="0x1" subaddress="0x1c" specific_type="4" size="1" frequency="3" poll="3"
onchange="0"/>
<reg name="myReg1" type="PCIe" specific_type="2" offset="0x0" size="1" poll="5"onchange="1"/>
<reg name="myReg2" type="PCIe" specific_type="4" offset="0x4" size="1" poll="5"onchange="1"/>
</node>
</hwtype>
```

User panel – Create device instantiation

- Use the HwTool as much as possible (fwHw_create, fwHw_setCommonSettings)

```
string device = "WorkshopBoard_Inst";
dyn_string settings = makeDynString("lbminidaq2-01", "0", "0");
fwHw_create("WorkshopBoard", device, TRUE);
fwHw_setCommonSettings(device, settings, true);
```

User panel – Read/Write registers

- Use the HwTool as much as possible (fwGbt_read, fwGbt_write)

```
main()
{
  dyn_dyn_char data;
  dyn_dyn_char mask;
  dyn_string   registers;
  dyn_int      status;
  int          i;

  dynAppend(registers,"FWGBT:Workshop_VLDB.VLDB_Master.sca.crb");
  dynAppend(data,fwHw_convertHexToByte("FE"));
  dynAppend(mask,fwHw_convertHexToByte("FF"));
  dynAppend(registers,"FWGBT:Workshop_VLDB.VLDB_Master.sca.crc");
  dynAppend(data,fwHw_convertHexToByte("FF"));
  dynAppend(mask,fwHw_convertHexToByte("FF"));
  dynAppend(registers,"FWGBT:Workshop_VLDB.VLDB_Master.sca.crd");
  dynAppend(data,fwHw_convertHexToByte("3F"));
  dynAppend(mask,fwHw_convertHexToByte("FF"));
  fwGbt_write(registers, data, mask, status);

}
```

User panel - Monitoring

- Use the HwTool as much as possible (fwGbt_startMonitoring, fwGbt_stopMonitoring)

```
main(mapping event)
{

  dyn_string registers;
  dyn_int refresh_rates;
  dyn_bool on_change;

  dynAppend(registers, "FWGBT:Workshop_VLDB.VLDB_Master.sca.cra");
  dynAppend(registers, "FWGBT:Workshop_VLDB.VLDB_Master.sca.crb");
  dynAppend(registers, "FWGBT:Workshop_VLDB.VLDB_Master.sca.crc");

  dynAppend(refresh_rates, 2);
  dynAppend(refresh_rates, 2);
  dynAppend(refresh_rates, 2);

  dynAppend(on_change, TRUE);
  dynAppend(on_change, TRUE);
  dynAppend(on_change, TRUE);
  fwGbt_startMonitoring(registers, refresh_rates, on_change);
  fwGbt_stopMonitoring(registers);
}
```

User panel - Connecting to your registers

- Use the HwTool as much as possible (dpConnect, fwGbt_convertHexTo….)

```
main(mapping event)
{
  string registers = "FWGBT:Workshop_VLDB.VLDB_Master.sca.cra.readings";
  dpConnect("myCallback", registers);
}

void myCallback(string dpName, dyn_char value){

  DebugTN(fwGbt_convertHexToDec(value));
}
```

Troubleshooting

A particular register is misbehaving!

- Check that you can read it with GbtClient. If you can, there might be something wrong with the register settings (or hwtool/Gbtserver, let us know). If you can not, the error message in the GbtClient is in general helpful as to what the problem should be. “Elink disconnected” might mean your Master GBT is not properly configured.
- If you cannot, then try to read it manually with the command line tools. If you can read/write it, then there is something being mishandled in GbtServer, otherwise this might be a firmware issue or an issue with your setup. Check your connections, try another link, make sure your firmware has a SOL40_SCA instantiated in the link your are using.
- Godspeed!

Command line tools

A set of tools that you can use to communicate directly with the SOL40_SCA without having to pass through the GbtServer (uses the same low level libraries as the GbtServer).

Usually they are named:

- <protocol>_test (eg. jtag_test): These implement the basic functions of the SCA, like accessing control registers and data register in the SCA itself
- <protocol>_scan, <protocol>_read (eg. jtag_scan, spi_scan): These implement the higher level functions such as perform a DR-SCAN in the JTAG line or a full SPI scan in the chain.
- dpconfig and configure_kintex7: Tools to manipulate front-end FPGAs through the JTAG line

These tools are distributed with the GbtServer rpm (--help for usage).

Remote FPGA configuration through SCA

KINTEX 7

- Supports Xilinx Series7 FPGAs
- Uses JTAG port
- Configuration time is very dependable on design and compression
- Implements only PROGRAM action so far
- Daisy chain is work in progress
- Partial reconfiguration will be needed

Microsemi

- Adaptation of DirectC tools
- Supports SmartFusion2 and IGLOO2
- Works for daisy chain without an issue
- Implements PROGRAM, VERIFY and ERASE actions
- Uses JTAG SCA port
- SPI programming in test/debug stage

Features now in place

FPGAs supported	**Ultrascale series 7 family**	**IGLOO2, SmartFusion2**
Protocol used	JTAG	JTAG (check), SPI (debug phase)
Timing	~35 secs for 3MB file	7 min for 3.5MB file
Daisy chain	YES	YES
Actions	PROGRAM	PROGRAM, ERASE, VERIFY, READ_ID, READ_INFO
Partial Reconfiguration	YES	NO

How to configure your FPGAs

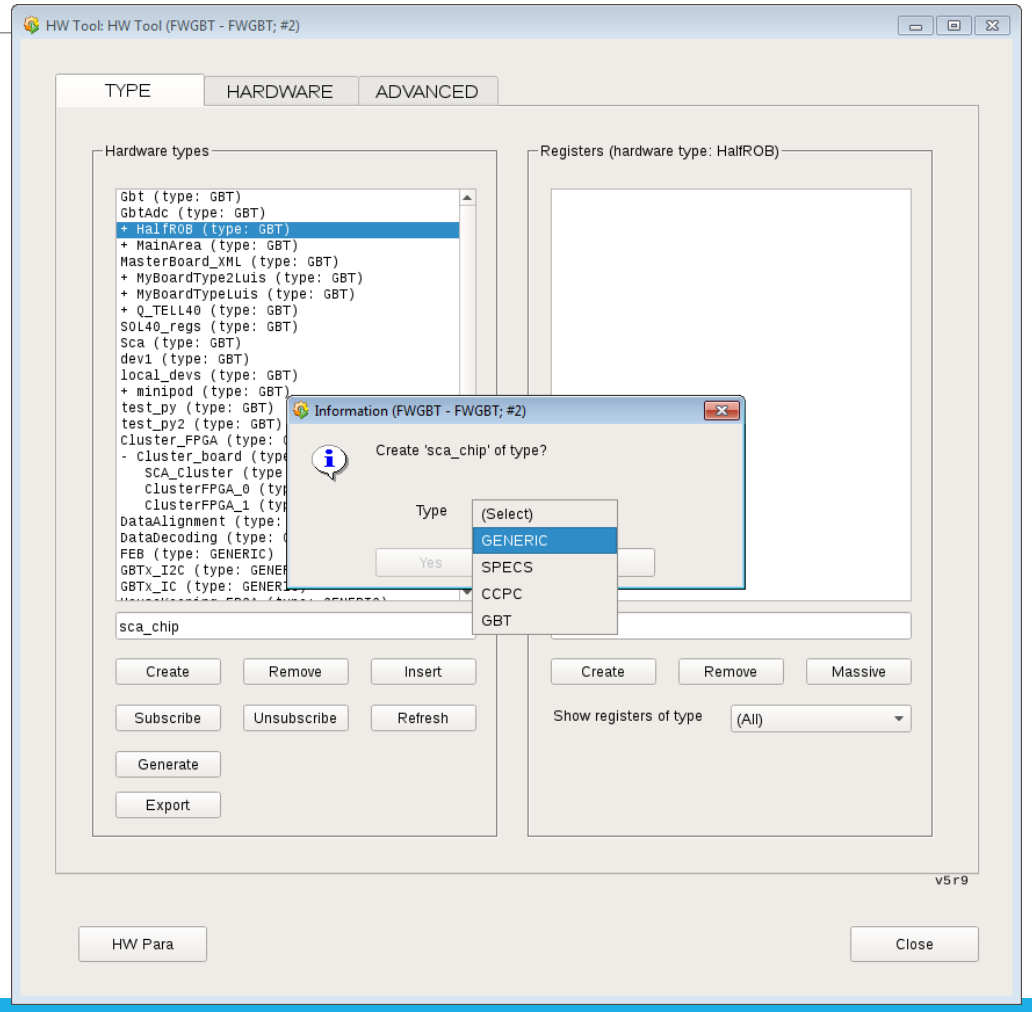
- In the GbtClient, using the FPGA tab (for testing)
- Using the fwGbt function:

```
int fwGbt_FPGAConfig(string pGbtPC, int pGbtId, int pScaId,
                     int pFrequency, int pRxEdge, int pTxEdge,
                     int pTrDir, int pIdle, int pChainLength,
                     int pDeviceIndex, int pRegLength, int pFpgaType,
                     string pFilename, int pPartial = FWGBT_FPGA_PROGRAM_FULL,
                     string pAction = "", int pTimeout = 0);
```

ANNEXES

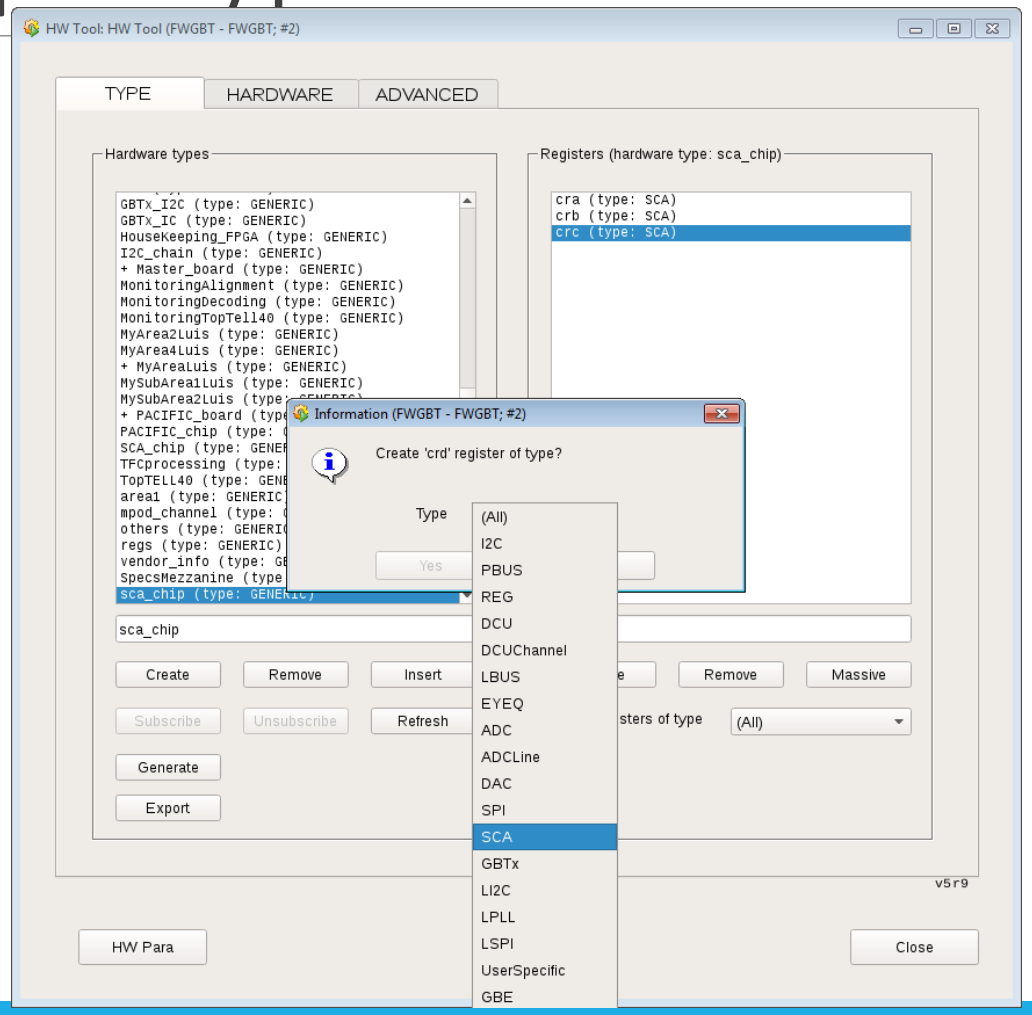
HwTool – Building your model

- In GEDI, go “LHCb Framework” -> “Hw” -> “Hw Tool”
- Choose a type name and “Create”, choose the type (typically GENERIC)



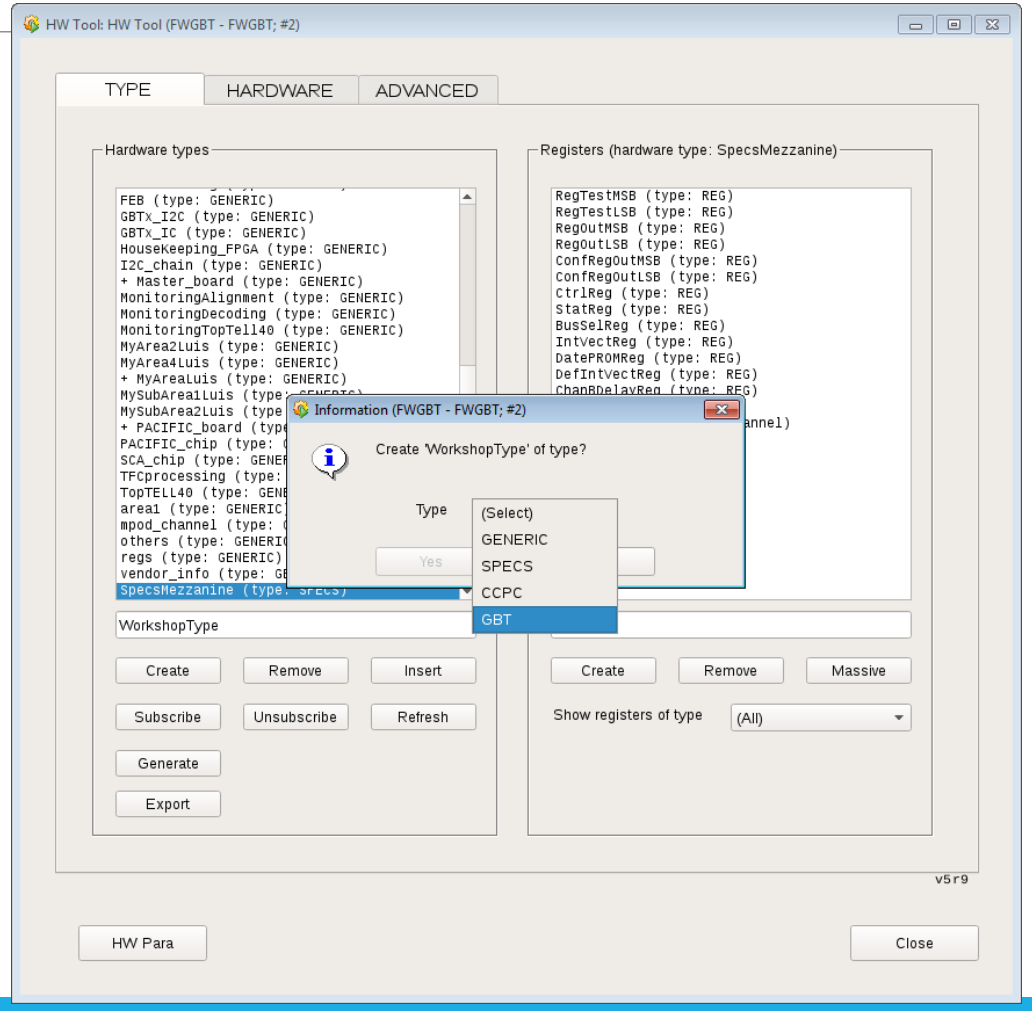
HwTool – Building a simple type

- In GEDI, go “LHCb Framework” -> “Hw” -> “Hw Tool”
- Choose a type name and “Create”, choose the type (typically GENERIC)
- Add registers to this type



HwTool – Building an hardware model

- Choose a type name and “Create”, choose the type



HwTool – Building an hardware model

- Choose a type name and “Create”, choose the type
- “Insert” type instantiations to this entity
- Continue until you have a final board that you want to instantiate in your system