FPGA MGT builder

A. Greshilov, Rice University
A. Madorsky, University of Florida/Physics
2020-04-29

Introduction

FPGA MGT builder is a set of software tools with the following functionality:

- Automatic generation of firmware structure with support for arbitrary MGT configurations. This includes:
  - Different bit rates and encodings in RX and TX parts of the same MGT
  - Using CPLL and QPLL as needed for each MGT, programmable separately for RX and TX parts (within the constraints of the particular FPGA architecture)
  - Automatic routing and assignment of available reference clocks for each MGT
  - Grouping MGTs into interfaces with programmable names and indexes, which makes using them in the top-level design much easier
  - Automatic generation of all constraints related to MGTs
    - Reference clock location, timing, and grouping
    - MGT location
    - User clock timing
  - The software is designed to make porting a project to different board design or FPGA an easy task.
  - Targeting lowest-latency serial links, by disabling RX and TX buffers
- Software framework:
  - Reads configuration settings and programs all DRP registers and port settings in each MGT and COMMON modules
  - Customizable reset procedures, with main functionality provided in the example design
  - Written in portable C++, can be adapted for nearly any system, including embedded processors
  - Does not need rework if the MGT configuration is changed
- Configuration sources:
  - All source configuration files are kept in Excel XLSX format
    - Makes working with them much easier
  - Data format is optimized for direct copying from Xilinx manuals and example source code, with minimal manual rework
  - A Python script is provided for exporting configuration files into plain text
- Currently supported:
  - System Verilog HDL
  - Linux
  - Xilinx 7-series FPGAs, GTH transceivers
    - Ultrascale support is coming soon

- Software needed:
    - Java run-time for running unmodified MGT firmware generator (provided as JAR archive)
    - Eclipse and Java compiler if you want to recompile the firmware generator
    - C++ compiler
    - Python 3 with `openpyxl` package
- Languages used:
    - Firmware generator: Java
    - Software framework: C++
    - Configuration exporter: Python

Getting started

The easiest way to familiarize yourself with the FPGA MGT builder is to use the example configuration provided with the software. The examples provide full support for the following FPGA:

- XC7VX690T-FFG1927

Step by step instructions:

1. Download the entire package from Github
2. Go into `examples` directory
3. There are three example projects provided. Each project is a directory. Make a copy of one of them, keeping internal directory structure.
4. Go inside the new project directory, `src` subdirectory
5. This directory contains all source files in Excel format
    a. `Protocols.xlsx` contains user's protocol settings
    b. `MTF7.xlsx` contains settings related to a particular board which is used in this project. A project can be used on multiple boards.
    c. `xc7vx690tffg1927-2_GTH.xlsx` file contains settings related to a particular FPGA and MGT family used in this project. A project can be used on multiple FPGAs and MGT families
6. At this time, you only need to modify protocols.xlsx. Open this file in LibreOffice Calc
7. Click on a tab named "EMTF_links". This is the list of all RX and TX links to be used in the design. Columns in that file are shown below:

Column number	**Description**
1	Link type, `tx` or `rx`
2	MGT X coordinate
3	MGT Y coordinate
4	Protocol name, must match one of the available protocols. See `protocols` directory for the available protocols.
5	QPLL or CPLL used (Q and C respectively)
6	Reference clock frequency, in MHz. May contain decimal point

7	Link group name, arbitrary. Will be used in the generated code as interface name
8	Link group index. Indexes within each group must start from 0, and run contiguously, without gaps.

8. Modify the Link table as needed. For example, try changing some of the protocol names, or MGT coordinates. Make sure to use valid coordinates and existing protocols.
9. When done, save the file.
10. Export the spreadsheet into text format:
    a. Go back into the top project directory, for example: `mgt_builder/examples/MPCX`
    b. Issue the following command: `python3 ../../python/xls_to_txt_openpyxl.py src/protocols.xlsx`
    c. This procedure automatically exports the contents of the XLSX file into text files and puts them into appropriate directories within the project.
11. In the project directory, find a file that has "`root`" in the name. This is your main configuration file (also exported from `protocols.xlsx`). For example, in MPCX project the file is named: `EMTF_MPCX_root.tab`
12. Generate firmware:
    `java -jar ../../java/mgt_builder.jar EMTF_MPCX_root.tab`
13. Visit "`sv`" directory and inspect the generated files. The list of generated files is shown below:

File name	**Description**
`mpcx_serial_io.sv`	Top-level module
`quad_module.sv`	Quad module
`mgt_module.sv`	MGT module
`common_module.sv`	COMMON module
`assign_mgt_rx.sv`	RX interface assignment module
`assign_mgt_tx.sv`	TX interface assignment module
`mgt_interfaces.sv`	Interface definitions. When importing into Vivado, mark this file as Verilog header
`xymap.sv`	X-Y coordinate map of the MGTs. When importing into Vivado, mark this file as Verilog header
`mpcx_serial_io.xdc`	Constraint file

14. The names of the generated files and the top SystemVerilog module are programmable via "root_config" tab in `protocols.xlsx` file. If you modifly any of them, don't forget to export XLSX file into text as shown in step 10.
15. Import the generated files into your Vivado design.
16. Also import files from `sv_manual` directory. This is a supporting logic for DRP and port access. Do not modify files in `sv_manual` directory.
17. In your Vivado project:
    a. MGT builder currently generates a place-holder for MMCM to manage the shared TX clocks. They need to be replaced with BUFGs as shown below:

Generated code example:
```
MPCX_mmcm MPCX_15_inst (.O(MPCX_15_mmcm_clk), .I(mgt_tx_if[0][0].txoutclk));
```
Reworked code example:
```
BUFG MPCX_15_inst (.O(MPCX_15_mmcm_clk), .I(mgt_tx_if[0][0].txoutclk));
```

        i. There may be several lines like this depending on your configuration.

b. Instantiate the generated top-level module in your design. Make sure to connect reference clocks and DRP bus signals.
c. Reference clocks should propagate to top-level module header unchanged.
d. DRP bus signals should be generated from your control interface. The generated firmware requires only 64KB of address space; this number does not depend on the number of MGTs in the design.
e. Instantiate TX and RX interfaces to be connected to the generated module. See top-level generated module declaration for details. Connect these interfaces to generated top-level module, and to your firmware module(s) that work with MGTs.

RX interface details

FPGA MGT builder is targeted for lowest serial link latency. RX logic is generated with the assumption that the RX buffers are disabled. Therefore, each RX interface comes with its own data clock, buffered via BUFH. The name of that clock is `rxoutclk`. The user is expected to take care of the clock domain crossing from `rxoutclk` of each MGT to the common fabric clock.

TX interface details

TX logic is also generated with the assumption that TX buffers are disabled. However, at this time the examples are set up for using TX buffers. This does not add a lot of latency (typically 2 XCLK periods). However, the work is ongoing to fully support disabling TX buffers.
As recommended by Xilinx manual, in the generated firmware the TX clock is shared between MGTs that:

- Are using the same protocol
- Are using the same reference clock input

These shared clocks are presented as outputs of the generated top-level module. The names for them are constructed as follows:

`MPCX_15_mmcm_clk`

Where “MPCX” is the protocol name, and “15” is the reference clock index. The user must make sure to use each TX clock with TX interfaces from corresponding MGTs. Identifying which MGT belongs to which TX clocks is easy:

- Open top-level generated module code
- Find a section with TX clock assignment (at the bottom)
- The comments at the end of each assignment line show interface name and MGT index

The user is expected to take care of the clock domain crossing from the common fabric clock to each of the TX clocks.

Creating new protocols

The easiest way to create a new protocol is to import the settings from an IP generated by Vivado. Step-by-step guide is below:

1. Open file protocols.xlsx in your project
2. Create two new tabs in it, one for MGT settings, one for COMMON module settings. Give them some arbitrary, easy to remember, names.
3. Create headers in each tab. These headers are showing where these tabs will be exported as files. Header format:

	A	B
1	// path	protocols/PROTOCOL_NAME
2	// file	“mgt_attributes.tab” for MGT, “common_attributes.tab” for COMMON
3	// columns	2

In this table, PROTOCOL_NAME is the name of your new protocol. File name cell should contain one of the two indicated names, unmodified. “A”, “B”, “1”, “2”, “3” markers are part of the spreadsheet frame, shown here for clarity.

4. Create a new IP core in Vivado, for a single MGT, with the exact settings that you’d like to have for your new protocol. If you are creating only TX or RX part of the protocol, you don’t have to worry about settings for the unused part (RX or TX, respectively). Include all service circuitry into the core, not into the example design.
5. Make sure to disable RX buffer for RX protocols, enable TX buffer for TX protocols. (Disabling TX buffers will be supported in future versions).
6. Open source code for the generated IP, by expanding the IP core in Vivado source browser.
7. Find the lowest-level wrappers for MGT and COMMON components.
8. In each of these two wrappers, find the instantiation of the MGT and COMMON library module. For 7-series devices, the library module names would be similar to GTHE2_CHANNEL (for MGT) and GTHE2_COMMON (for COMMON).
9. Copy settings for all attributes and ports from the wrappers into corresponding tabs in the protocols.xlsx file. For 7-series example designs, it’s as easy as copying and pasting directly from Verilog source file, splitting the text on the following delimiters:

   ```
   TAB Space . ( ) "
   ```

   Keep the standard Xilinx syntax like “tied_to_vcc” unchanged, FPGA MGT builder recognizes it.
10. **See additional importing instructions in the example protocol.xlsx files, Instructions tab.**

11. Create one more tab in the spreadsheet, for protocol constraints. The contents of that tab should be as shown below:

	A	B
1	// path	protocols/PROTOCOL_NAME
2	// file	protocol_config.tab
3	// columns	2
4	rxoutclk_constr	6.25
5	txoutclk_constr	6.25

First 3 lines are header, lines 4 and 5 are RXOUTCLK and TXOUTCLK period timing constraints, in ns. These constrains should be calculated from your protocol settings.
Example:
Line rate: 3.2 Gbps
Bits transferred in one frame: 20
RXOUTCLK, TXOUTCLK frequency: 3.2G/20 = 160 MHz, period = 6.25 ns

12. Your new protocol is now ready to use.

Using and adapting C++ software for your design

TBD