Scotty Cowling, WA2DFI

PO Box 26843, Tempe, AZ 85285: **scotty@tonks.com**

Hands-On-SDR

The author explains using the FPGA with SDR designs.

In this installment we will continue onwards and upwards into more inner workings of the Field Programmable Gate Array or FPGA used in many of our SDR designs. This column relies heavily on what I covered in my Mar/Apr 2015 and Jan/Feb 2016 columns.[1,2] If you have not read at least the most recent one, please take a minute and do a quick review. It has been a while since we covered some of this material, so we will start with a quick review here.

In the Mar/Apr 2015 column, we covered getting the free tools set up and showed you how to compile and run an example design. In the Jan/Feb 2016 issue, we covered porting of open-source FPGA code to run on Arrow's BeMicroCVA9 FPGA development board used in the Hermes-Lite and IQ2 Software Defined Radios.

Many revisions have been made to the Hermes software that we used as a starting point in my last column. As a review, I will quickly cover the steps necessary to port the newest version of the Hermes FPGA code to the BeMicroCVA9. Please refer to my last column for more detail or if you need a refresher.

Once again, I want to thank Phil Harman, VK6PH, for his work in updating this FPGA code. It is truly a Herculean task!

What Do We Need to Get Started?

As with each of these columns, I always try to define what you need in the way of knowledge and equipment to get the most out of the "Hands On SDR experience". You will need a basic working knowledge of the Verilog hardware description language. Once again, my assumption is that the existing code is working, and we will try not to introduce any new bugs as we port to the new device. As we did last time, we are *targeting* a new device, not *designing code* from scratch.

For hardware, you will need a BeMicroCVA9 development kit.[3] To actually run the code that we are going to compile in this column, you will also need an HF2 board (to receive), or both HF2 and TX2 boards (to transceive).[4,5] As a lower-performance (and less expensive) alternative, you can use an HF1 or Hermes-Lite, but you will need to make other modifications to the code if you go that route.[6,7] I believe that the Hermes-Lite group has ported their firmware to the BeMicroCVA9. After wading through this column, you should be expert enough to compile their source and run it on the BeMicroCVA9. Even if you do not have the hardware, you can still follow along with the text and learn about porting FPGA code to new devices.

For tools, we will still need *two* versions of Altera's Quartus FPGA design software to complete the porting work. The first version is Quartus II version 13.1, which is the version that was used to create the code that we are going to port. The second version is the latest (and newly released as of this writing), version 16.0. It is now called Quartus Prime Lite, and requires a 64-bit operating system. You will need 64-bit Windows XP, Windows 7 or later or 64-bit Linux in order to run this new version. All of the information from my Mar/Apr column applies to both Quartus versions. Before you continue, you will need to download and install both of the free web versions (Quartus II version 13.1 and Quartus Prime Lite 16.0) from the Altera web site.[8] To save some download time, you only need to download Cyclone III and Cyclone V device support for Quartus II version 13.1, and only Cyclone V device support for Quartus Prime Lite version 16.0.

Why Two Quartus Versions?

The conditions that required us to use two versions of the design software still exist, even with the release of the new Quartus Prime Lite 16.0. I explained this in my last column, but it is important enough to bear repetition here. The explanation is tied to the capabilities of each Quartus version and the FPGA part that we are migrating *from* as well as the part we are migrating *to*. The Hermes code targets the Cyclone III EP3C25Q240C8 (our *from* part number), while the BeMicroCVA9 uses a Cyclone V 5CEFA9F23C8 (our *to* part number). Quartus II version 13.1 supports all Cyclone III parts and some of the Cyclone V parts, but unfortunately not our *to* part number. Quartus Prime Lite version 16.0 supports

Table 1.
Clock name changes in the Hermes.sdc file.

Line(s)	*original name in **Hermes.sdc** file*
19, 154	PHY_CLK125
33, 143	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[0]
34, 144, 184	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[1]
35, 145, 184	PLL_IF_inst|altpll_component|auto_generated|pll1|clk[2]
37, 157	network_inst|rgmii_send_inst|tx_pll_inst|altpll_component|auto_generated|pll1|clk[2]
39, 158, 171	network_inst|rgmii_send_inst|tx_pll_inst|altpll_component|auto_generated|pll1|clk[4]
48, 156	network_inst|rgmii_send_inst|tx_pll_inst|altpll_component|auto_generated|pll1|clk[1]
155, 169, 171, 193	network_inst|rgmii_send_inst|tx_pll_inst|altpll_component|auto_generated|pll1|clk[0]
161	OSC_10MHZ PLL2_inst|altpll_component|auto_generated|pll1|clk[0]

all Cyclone V parts (including our *to* part number), but no Cyclone III parts at all! The easiest way to migrate to the new part and new Quartus version is to change part families first and then upgrade to the latest version of Quartus as a separate operation. Here is the flow of part numbers and Quartus versions that we will use: 3C25 with v13.1 → 5CEFA7 with v13.1 → 5CEFA7 with v16.0 → 5CEFA9 with v16.0.

Notice that Quartus II version 13.1 does not support our 5CEFA9F23C8 part, so we pick a dummy part (5CEFA7F23C8) that it does support just to get us into the Cyclone V family. After we migrate to Quartus Prime Lite version 16.0, we will pick our final, correct 5CEFA9F23C8 target. Also notice that in the flow above, we only change *one* item in each step: either the part number or the Quartus version, but never both.

FPGA Code Porting Tasks

We covered these steps last time, but here they are again:

- Open design in original Quartus version
- Update wizard-generated modules
- Add code to hook in new signals and remove unused old signals
- Add new location properties
- Update SDC timing constraints file with new signals and remove old signals
- Compile-debug-repeat.

I explained the first four steps in detail last time, so I will focus on the last two this time around. That doesn't mean that I will leave you completely on your own, just don't expect as much detail as last time. We want save room to do some *new* things! Unfortunately, we have some *old* things to get out of the way before we can move on to the new.

Open and Compile the Design

To get a copy of the FPGA source code, download a copy of the Quartus archive from the SDRstick SVN webserver.[9] Open the archive in Quartus II version 13.1 and fire off a trial compile right off the bat. This will tell you if you have everything set up correctly. You should get a bunch of warnings from Quartus (I got 400!), but no errors. As usual, if Quartus reports errors, you must fix them before you can continue.

The cleanup step that we had to perform last time has already been done as part of the many upgrades that have been done since we last looked at the code.

Update Wizard-generated Modules

The Hermes design uses four PLLs, seven FIFO memories, four ROM memories, one RAM memory, one multiplier and three other functions for a total of 20 Wizard generated modules. Check the IP Components tab of the Project Navigator to see a list of IP components and version numbers. Each of these modules must be updated first to the Cyclone V family under Quartus II version 13.1 before we can open them in Quartus Prime Lite version 16.0.

Move the design to the Cyclone V family and remove all location assignments. We will add the new (and different) location assignments for the new FPGA part number later. Select the **5CEFA7F23C8** part. Note that this is not the final part, but an intermediate one that we must pick due to the vagarities of the Quartus software. And we are still using Quartus II version 13.1. We will fix both of these problems after we finish updating the wizard-generated modules.

Update the 20 wizard-generated modules, taking care with the PLLs and the **firromH** module. Remember that Cyclone V PLLs are different from Cyclone III PLLs, so you must create new ones and replace the old ones with the new ones. Close Quartus and re-open the project in Quartus Prime Lite version 16.0. The new version of Quartus will ask you if it should overwrite the database with the new format. You can safely answer **Yes**. Change the part number to 5CEFA9F23C8 and run a compile to see if we broke anything. Now we are using version 16.0 with the correct FPGA part number. We are almost done!

Add and Remove Code and Signals

The next step in our 6-step program is to match up the old design (Hermes) signals with the new design (CVA9) signals. Remember to account for every one of the Hermes signals, as well as every one of the new design pins (CVA9) by either ignoring it, adding code to support it or just connecting it to its counterpart from the old design. I have created a file for you containing a table of all of the signal names in the design to help make the changes. This **Hermes_1_May_to_IQ2_pins** table will tell us which pins map directly onto new pins and which do not.[10] I will not revisit the changes covered in my last column; please refer back to it to make the changes (see Note 2).

Add New Location Properties

Now it is time to add the new location properties back in to replace the old ones that we deleted when we changed part numbers. Again, I have created a file for you to save you the effort of typing all those lines into the script file. You can download it from the SDRstick SVN webserver.[11]

To run the script, place the file in your top directory (that is, the directory that contains your **Hermes.qsf** file and all of your Verilog source files). Now add it to your project using **<Project> <Add/Remove Files in Project…>**. Under **<Tools> <Tcl Scripts…>**, select the file and click **Run**. All of your pin locations from the script file have now been added. If you want to check your new assignments (you should believe me by now) you can open the Assignment Editor from (where else) the **<Assignments> <Assignment Editor>** menu. You should see all of your new **Location** assignments listed. Run a compile to make sure things are as they should be.

Wow, all that work just to get to the same point that we were at the end of the last column! Well, not quite... This time we started with FPGA code that is many revisions better than the version that we started with last time, and we are now using the latest and greatest version of the Altera tools (Quartus Prime Lite 16.0). And best of all, we have proven that we have learned enough to do it over and over again. Next time, no peeking at the previous column!

*new name in **Hermes.sdc** file*

```
DDR3_CLK_50MHZ
PLL_IF_inst|pll_if_new_inst|altera_pll_i|cyclonev_pll|counter[0].output_counter|divclk
PLL_IF_inst|pll_if_new_inst|altera_pll_i|cyclonev_pll|counter[1].output_counter|divclk
PLL_IF_inst|pll_if_new_inst|altera_pll_i|cyclonev_pll|counter[3].output_counter|divclk
network_inst|rgmii_send_inst|tx_pll_inst|tx_pll_new_inst|altera_pll_i|cyclonev_pll|counter[2].output_counter|divclk
network_inst|rgmii_send_inst|tx_pll_inst|tx_pll_new_inst|altera_pll_i|cyclonev_pll|counter[3].output_counter|divclk
network_inst|rgmii_send_inst|tx_pll_inst|tx_pll_new_inst|altera_pll_i|cyclonev_pll|counter[1].output_counter|divclk
network_inst|rgmii_send_inst|tx_pll_inst|tx_pll_new_inst|altera_pll_i|cyclonev_pll|counter[0].output_counter|divclk
PLL2_inst|c10_pll_new_inst|altera_pll_i|cyclonev_pll|counter[1].output_counter|divclk
```

Update SDC Timing Constraints

Now we will update the **Hermes.sdc** timing constraints file line by line to remove constraints for signals that we have removed, add (or expand existing) constraints for new signals and update constraints for anything that we changed. This is where we left off last time, so it is time to do it now.

Most of the changes to the **Hermes.sdc** file are due to the changes that we made to the PLLs. The SDC file refers to the PLL pins by name, and remember that we changed some of them. We have to fix the names in the SDC file so that the timing analyzer can match them up with the design files. I have listed the changed names in Table 1. The first entry is not a PLL change, but a clock pin name change. Remember that there is no PHY_CLK125 clock from the Ethernet PHY chip on the BeMicroCVA9. We changed that to a 50 MHz clock (from an external oscillator). The only place that this 125 MHz clock was used was as a reference clock input to the tx_pll. When we created tx_pll_new, we simply changed the PLL programming a bit so that it uses a 50MHz reference rather than the original 125MHz reference. On line 19 of the **Hermes.sdc** file, change **PHY_CLK125** to **DDR3_CLK_50MHZ** in both places it appears, then change the **8.000** after -"period" to **20.000**. Why? Because this number represents the clock period in ns; 8ns period is 125 MHz and 20 ns period is 50 MHz. Make the name changes shown on each line in Table 1. Note that some lines require multiple changes.

Next, we want to remove references to any signal that we removed. Rather than remove a line, comment it out by placing an octothorpe (# symbol) in the first column of the line[12]. Affected lines (and signal names) are 88 (SO), 94 (ADCMISO), 120 (MOSI, nCS), 123 (CMODE only), 126 (J15_5, J15_6, SPI_SDO), 129 (CS, SCK, SI), 132 (ADCMOSI, nADCCS), 196 (SSCK, ADCCLK, SPI_SCK only), 205 (USEROUT* only), 208 (ANT_TUNE, IO4-IO8 only). Note that the lines that I have marked "only" cannot be commented out, since we are only removing the reference to the listed signals. Other signals listed on the same line must remain, so just delete the signal(s) that I have indicated above and leave the rest alone. Since we removed all of the pins associated with the EEPROM (since the CVA9 does not have one), we can comment out line 211. Since we commented out lines 94, 126 and 132, **data_clk2** is no longer used; we can comment out line 54 and remove line 147. (Leave just the "\" on line 147 to preserve the line numbering.) The last thing we will do is comment out lines 70 and 104 to eliminate unnecessary timing constraints on the ASMI block, which we upgraded to a Cyclone V version.

This should result in a Hermes.sdc file that generates no warnings. To check to see if we missed anything, open TimeQuest by clicking on **<TimeQuest Timing Analyzer>** under the **<Tools>** menu. Once TimeQuest opens, double click on **Update Timing Netlist** in the Tasks pane on the left side of the screen. This will cause all three tasks listed under Netlist Setup to run: **Create Timing Netlist**, **Read SDC File** and **Update Timing Netlist**. All three of these lines should turn green, and a check mark should appear next to each of them (see Figure 1). Most importantly, though, is that any warnings will appear in the Console pane across the bottom of the screen. If you see any warnings, then TimeQuest is not happy with your **Hermes.sdc** file and you should make corrections before proceeding. If you want to see what a warning looks like, go back to the Hermes.sdc file and undo one of the fixes that you just put in. (As an example, un-comment out line 70.) Save the SDC file, return to the TimeQuest screen (or re-open TimeQuest) and this time click on **Reset Design** before you click on **Update Timing Netlist**. Doing this tells TimeQuest to re-run the three Netlist Setup tasks from scratch, so you get a fresh read-in of the SDC file. Note that you do not need to recompile the design to do this. The design hasn't changed; we are merely checking the design against different timing constraints to see if it meets them. If you actually changed a timing parameter (such as a clock period or an input delay), you would have to recompile your design so that Quartus could optimize the routing to try and meet your new constraint.

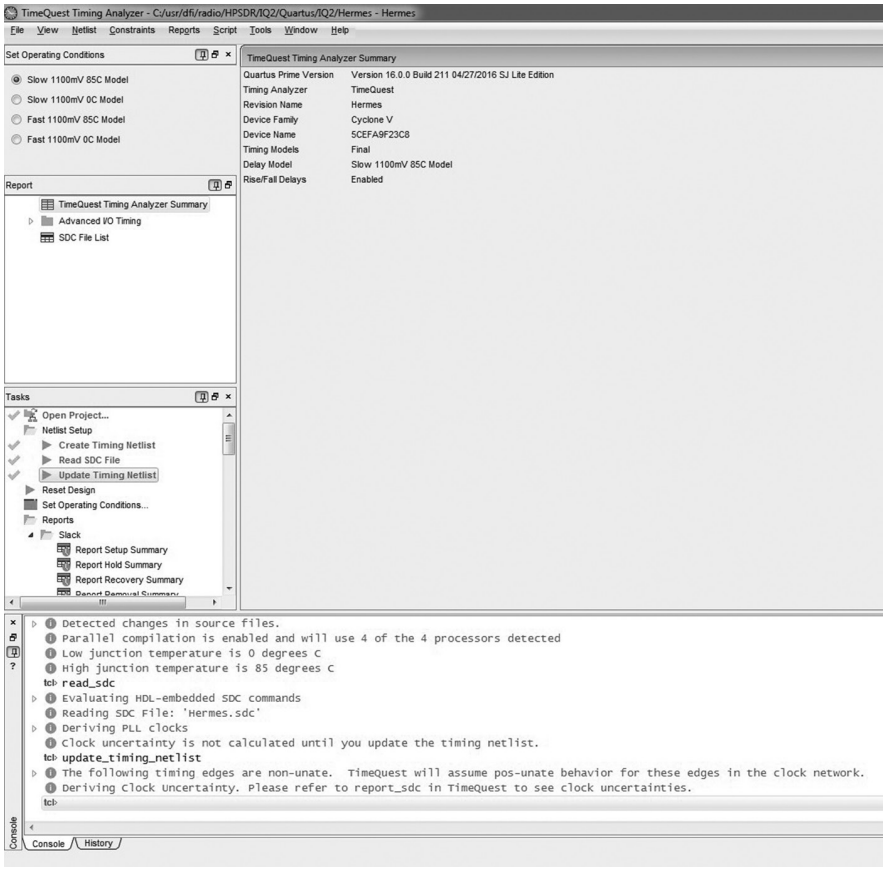

Figure 1 — TimeQuest timing analyzer.

Compile-Debug-Repeat

The last thing we will do this month is to wade through some of the warnings that Quartus generates to get a feel for which ones can be safely ignored and which ones you should fix. My last compile generated 0 errors and 140 warnings. Your numbers may be slightly different, but not *too* different. This seems like an awful lot of warnings, doesn't it? After we review some (or most) of these warnings and their causes, you will see that, in fact, it really isn't that many. Quartus "warns" you about many things that you either can't do anything about because they are generated by internal code that you cannot edit or are simply unimportant, such as a size mismatch in an assignment statement. Quartus also warns you about things that really are problems, just not fatal ones. For example, suppose we forgot to

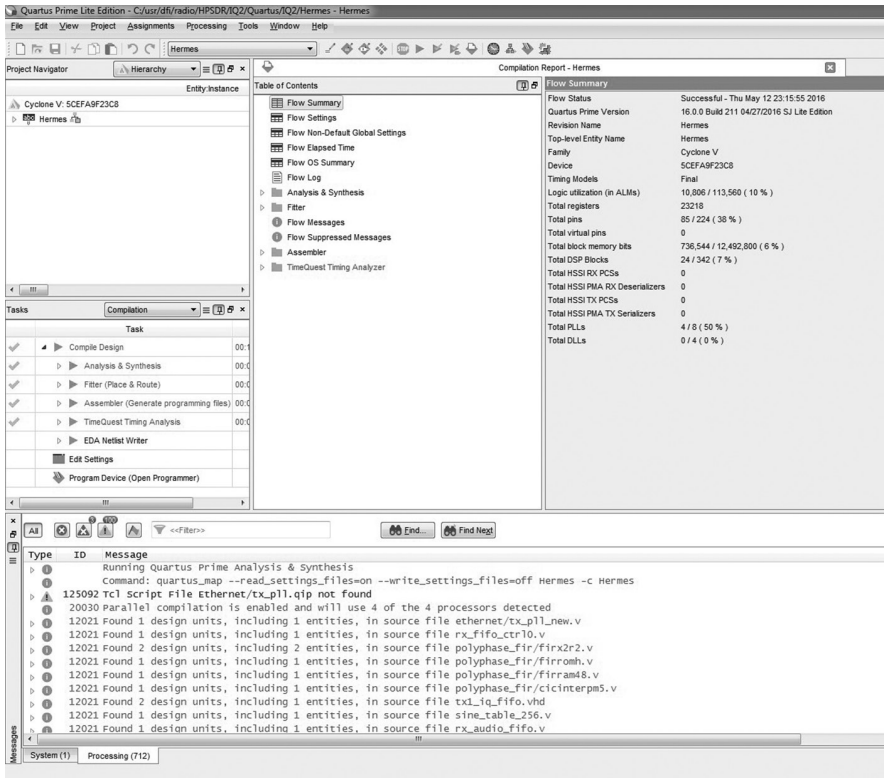

Figure 2 — Quartus Prime Lite 16.0 screen showing a warning in the message pane.

connect a signal to an I/O pin on the part. Quartus will remove all of the unused logic that connects to that signal. Maybe that is OK if you did it intentionally. If it was an oversight (we won't say *error*), you will be grateful that Quartus warns you that it removed logic and why it did so. The bottom line is that you must look at every warning to determine if it is important enough for you to investigate its cause. Since there are typically many warnings, you must be able to quickly assess the importance of each one. This takes skill, and skill comes through experience. So let's get some experience now.

Start a new compile and let it run to completion. After it finishes, scroll up in the messages window (the full width pane across the bottom of the Quartus window) to the first line that appears in blue. Warning messages are in blue and information messages are in green. Error messages are in red and will generally stop the compilation prematurely (you should not see any of these). You may or may not get the same messages that I get or in the same order that I get them. It will depend on the changes that you made versus the ones that I made, and in what order you made them as well as any mistakes that I made that you did not (or vice-versa). It will also depend to some extent on what options you have set in your project. Most warning messages will take you to the source of the warning if you double-click on the warning text, but not all of them will do this. Finding the source of the warning is the first step. Correcting it is the second, unless you decide that it is unimportant and can remain.

The first blue line that I encounter is (see Figure 2): **125092 Tcl Script File Ethernet/tx_pll.qip not found**

This is interesting, since it refers to an old PLL that I removed from the project, tx_pll. Click the triangle in column 1 to expand the warning and get more information. Unfortunately, this is one of those warnings that you cannot click on, so we have to figure it out for ourselves. The second line says: **125063 set_global_assignment -name QIP_FILE Ethernet/tx_pll.qip**

This is an assignment present in the Hermes.qsf file. How do I know this? From experience. How can you come to know this? Google! Paste **set_global_assignment** into Google search and the first hit explains more than you ever wanted to know. Go ahead, try it. The Quartus help page that Google points you to explains what the command does, its syntax, and so on. But all we need to know is *where it is located*, so we can remove it. It is in the project's Quartus Settings File, or **Hermes.qsf**. We must be especially careful when modifying the qsf file; it is kind of like editing your Windows registry. You can damage you project beyond repair if you edit this file with wild abandon. So here are three rules to follow to keep you project safe.

1. ***NEVER*** edit the qsf file while Quartus is open.

2. Always make a backup before opening

the file in a text editor

3. Use a *text editor* (like notepad) not a *word processor* to make changes

Notice that in #1, the word never is in bold, underlined italics. Quartus reads this file in, modifies this internal copy and then writes it out upon exit. If you change it while Quartus has it open, you are wasting your time, and just asking for trouble. So, after all that, exit Quartus, open **Hermes.qsf** in your favorite text editor, find the offending line and delete it. Save the qsf file (remember: text-only format) and exit your text editor. Re-open the project in Quartus (hint: use **<Recent Projects>** on the **<File>** menu). But wait, all of my messages are gone! Don't panic, they were saved just for you. Under the **<Processing>** menu, click **<Compilation Report>**, or just type <ctrl>R if you are lazy like me. When the report window opens, look in the left pane for **Flow Messages** and click on it. Like magic, all of your messages are back, although in a different window. (You sure are being picky!) On to the next warning! **10858 Verilog HDL warning at Hermes.v(1068): object frequency_change used but never assigned**

This is one that you can double-click, so go ahead and do it. Quartus automatically opens the **Hermes.v** file and highlights the offending line. Search through the file (use <ctrl>F) to see where the variable **frequency_change** is used. Note that it is passed to the CC_encoder module using the same name, so open **CC_encoder.v** and search for it there. Note that it is an input to CC_Encoder and used on line 108, but it is never set to a value anywhere. This is what Quartus is complaining about: shouldn't you set a variable to a value before you use it? Well, yes, but... If you choose to ignore this warning, you will get whatever the default value for the variable **frequency_change** is. Go back to your Flow Messages window, and it tells you what value it will use. Darn clever, this compiler. This is not fatal, so we will opt to come back and fix it later. Next! **10034 Output port "outclk_2" at PLL_IF_new_0002.v(17) has no driver**

When you double-click on this one, Quartus takes you to the **PLL_IF_new_002.v** file and highlights line 17. But wait, we didn't create this file, the Wizard did. This is one of those cases where we just leave it alone and live with the warning. There are lots of these "has no driver" warnings, and they all point to Wizard-generated files. We can ignore all of them. Next! **10230 Verilog HDL assignment warning at sdr_send.v(118): truncated value with size 32 to match size of target (8)**

This kind of warning is very common. It occurs whenever we try to assign a value represented in a certain bit width to a variable of a different width. Double click on the warning to see line 118 in **sdr_send.v**. The parameter **NR** has a width of 32 bits, while the variable **number_RX** is only 8 bits wide. Quartus tells us exactly what it is going to do: truncate (i.e., discard) the 24 upper bits of **NR** and use just the bottom 8 to set **number_RX**. Since I would have to figure out how to define an 8-bit parameter, and the result is what I wanted anyway, I don't have to fix this one either. On to the next. **12030 Port "extclk" on the entity instantiation of "cyclonev_pll" is connected to a signal of width 1. The formal width of the signal in the module is 2. The extra bits will be left dangling without any fan-out logic.**

If you double click on this one, you see that it is a Wizard generated warning, so we can't really fix it. **12020 Port "ordered port 0" on the entity instantiation of "fir3" is connected to a signal of width 32. The formal width of the signal in the module is 1. The extra bits will be ignored.**

This looks like the last one, but it points to **receiver2.v**, which is one of our files. This is like the truncated value warning. On line 144 of **receiver2.v**, the first value inside the parentheses is a zero. If you look at the file **firx2r2.v**, you will see that this corresponds to this input signal **reset**. The variable is one bit wide, but the default width of a number is 32 bits wide. Now since the number is zero in this case, it doesn't much matter. A better way would be to define the zero as a 1-bit constant (instead of 32-bits) like this: **1'b0**.

There are many more warnings than I have space to cover, but there is one more important one: **171167 Found invalid Fitter assignments. See the Ignored Assignments panel in the Fitter Compilation Report for more information.**

This usually means there are invalid fitter assignments in the qsf file that should be fixed or removed. To get to the Fitter Compilation Report, in the left pane of the compilation report click on the triangle next to **Fitter** to expand it, and then click on **Ignored Assignments**. Now you see a table (containing only one line) that shows you the name of the signal (**PHY_CLK125** in this case) and where it is located (the qsf file in this case). You already know how to do this: close Quartus, backup **Hermes.qsf**, open **Hermes.qsf** and remove the offending line, save the file, reopen the project in Quartus.

Hopefully this exercise has given you a better feel for Quartus and what its capabilities are along with the confidence to jump in and get your feet wet. The final step, of course, is to recompile the project, see *fewer* warnings than before, then load the compiled programming file into the BeMicroCVA9 and test it to make sure that it works. I will cover how to load and run the code on real hardware in my next column. An updated Quartus archive containing all of the changes that we have made is available on the SDRstick SVN webserver.[13]

What's Next?

Remember that the openHPSDR project is open source, and the Apache Labs Anan series of transceivers are all powered by open-source FPGA firmware. Each openHPSDR board has an on-board FPGA and Verilog code to match. All of it is available from the openHPSDR repository[14]. Try your hand at some FPGA coding, now that you see how easy it is! The tools that you have used today are the very same tools that the developers use when they write or update the code.

Source code and reference files for this article are on the **www.arrl.org/QEXfiles** web page.

As always, please drop me an e-mail if you have any suggestions for topics you would like to see covered in future Hands-On-SDR columns or even just to let me know whether or not you found this discussion useful.

Notes

[1]Scotty Cowling, WA2DFI, "Hands On SDR", QEX, Mar/Apr 2015, pp 9-19.

[2]Scotty Cowling, WA2DFI, "Hands On SDR", QEX, Jan/Feb 2016, pp 28-34.

[3]BeMicroCVA9 from Arrow Electronics: **arrow.com/en/products/bemicrocva9/arrow-development-tools**

[4]UDPSDR-HF2 from Arrow Electronics: **arrow.com/en/products/udpsdr-hf2/arrow-development-tools**

[5]UDPSDR-TX2 from Arrow Electronics: **arrow.com/en/products/udpsdr-tx2/arrow-development-tools**

[6]UDPSDR-HF1 from Arrow Electronics: **arrow.com/en/products/udpsdr-hf1/arrow-development-tools**

[7]Hermes-Lite wiki: **github.com/softerhardware/Hermes-Lite/wiki**

[8]Free Altera Web Edition software: **dl.altera.com/?edition=web**

[9]The source code is available from the SDRstick SVN at **svn.sdrstick.com** under the <sdrstick-release/BeMicroCV-A9/Hermes-HF2-Port/firmware/source> directory. The file name is <Hermes_1_May.qar>

[10]The cross reference of Hermes to IQ2 pins is available from the SDRstick SVN in the same directory as above. The file name is <Hermes_1_May_to_IQ2_pins.pdf>

[11]The pin location Tcl script file is available from the SDRstick SVN in the same directory as above. The file name is <Hermes_1_May_map_pins.tcl>

[12]Yes, a # symbol, commonly known as a pound sign is called an octothorpe. See **en.wiktionary.org/wiki/octothorpe**

[13]Source code containing all of the changes outlined in this column is available from the SDRstick SVN at **svn.sdrstick.com** under the <sdrstick-release/BeMicroCV-A9/Hermes-HF2-Port/firmware/source> directory. The file name is <Hermes_1_May_ported.qar>

[14]For HPSDR firmware, look in the TAPR repository **svn.tapr.org** in <main/trunk>