Scotty Cowling, WA2DFI

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

Hands-On-SDR

In this installment, we move back toward the basics (I didn't say simple!) and delve a bit more into the inner workings of the magical, mystical field programmable gate array, or FPGA. This column relies heavily on what I covered in my Mar/Apr 2015 column.[1] If you have not read that, or can't remember reading it, now would be a good time for a quick review. Since I can't remember writing it, I need to take a short break and read it again myself…

In the Mar/Apr 2015 column, I showed you how to set up an FPGA coding environment with free development tools, walked you through the code of an SDR design example, showed you how to compile the example code and run it on real hardware. We did cover some SDR theory, but we took much of the background as a given and instead focused on how we implemented functions *inside* the FPGA.

This time we will be taking the open-source code written for a variant of the high-performance software-defined radio (HPSDR) Hermes single-board transceiver (specifically the Apache Labs Anan-10e) and port it to the BeMicroCVA9 development board from Arrow Electronics. This board is used in the Hermes Lite project as well as in the IQ2 transceiver and is also compatible with the SDRstick HF1, HF2 and TX2 RF boards. I am going to focus on the HF2 receiver and TX2 transmitter boards, but I will include enough information for you to port the Hermes code to almost any compatible RF front-end board. Given the price and performance of the BeMicroCVA9, I expect that a bevy of hardware designs will surface once the word gets out. So let's start getting the word out!

We owe many thanks to Phil Harman, VK6PH, Kirk Weedman, KD7IRS, and Alex Shovkoplyas, VE3NEA, who wrote the original code that we will use as a starting point. As you look through the code, I think you will be grateful for their work. Without their significant efforts, we would have to write all of this complicated code ourselves!

What Do We Need to Get Started?

As with each of these columns, limited space begs the questions: "What do I need to know?" and "What equipment do I need?"

You will need a basic working knowledge of the Verilog hardware description language. If you followed my Mar/Apr column, you are prepared enough. We will not be deep diving into the intricacies of the code, since we are just porting the code to a new device. 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. Of course, my assumption may prove to be false, but that is a topic for another day: debugging FPGA code!

For hardware, you will need a BeMicroCVA9 development kit.[2] 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).[3, 4] As a lower-performance (and less expensive) alternative, you can use an HF1 or Hermes-Lite board, but you will need to make other modifications to the code if you go that route.[5, 6] 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 code 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.

Like my Mar/Apr column, you will need some Verilog programming knowledge, but SDR knowledge in general is not required. We are *targeting* a new device, not *designing code* from scratch.

For design software, there is good news and bad news. The good news is that Altera offers their *Quartus II* FPGA design software as a free download from the Internet for the FPGAs in their Cyclone® family of parts. Both *Linux* and *Windows* versions are available. We will need *two* versions of the *Quartus II* 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 (as of this writing), version 15.0. The bad news is that *Quartus II* version 15.0 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 *Quartus II* versions (13.1 and 15.0) from the Altera web site.[7] 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 II* version 15.0.

Why Two *Quartus* Versions?

Before we get down to the meat and potatoes, I need to explain some problems that we face that are unique to our task. You might ask "Why do we need two versions of *Quartus*?" The answer 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 II* version 15.0 supports all Cyclone V parts (including our *to* part number), but no Cyclone III parts at all!

While it is certainly possible to migrate *Quartus* versions and part families in one step (which was my original intent for this column), doing it that way is difficult. We will follow an easier course by changing part families first and then upgrading to the latest version of *Quartus*. If the complexity of this method frustrates you, try to remember that this is the *easy* way. To paraphrase a common saying: "There are only two ways to do this. If you don't like this one, you for sure won't like the other one!" Here is the flow of part numbers and *Quartus* versions that we will use:

3C25 with v13.1 → 5CEFA7 with v13.1

[1]Notes appear on page 34

→ 5CEFA7 with v15.0 → 5CEFA9 with v15.0

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 II* version 15.0, 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

Now that we understand the mess that we have gotten ourselves into, here is an outline of the WA2DFI 6-step program to successful FPGA code porting:

1) Open the design in the original *Quartus* version.

2) Update the wizard-generated modules.

3) Add code to hook in new signals and remove unused old signals.

4) Add new location properties.

5) Update the SDC timing constraints file with new signals, and remove old signals.

6) Compile-debug-repeat.

While none of these steps is fraught with peril, some are a bit more involved than others. Let's look at each step in more detail.

Open the Design

To get a copy of the FPGA source code, download a copy of the *Quartus* archive from the SDRstick SVN webserver.[8] The archive not only contains the source files (with a .v extension), but the pin assignment file (.qsf extension), timing constraints file (.sdc extension) and many other files needed to successfully compile the complete project.

Once you have downloaded the archive file, start the *Quartus II* version 13.1 software and click on **<file><open project…>**. Navigate to the .qar file that you downloaded and click on it. From the dialog box that opens, select the destination folder (usually the default is good) and click **OK**. *Quartus* will extract all of the files from the archive and set up the project, all ready to go. I recommend that you fire off a trial compile now (yes, right now!) with no changes. This will tell you if you have everything set up correctly. The compile button is the small right-facing triangle on the toolbar. If you prefer menus, the **<Start Compilation>** button is also under the **<Processing>** menu. You should get a bunch of warnings from *Quartus*, but no errors. If *Quartus* reports errors, you must fix them before you can continue.

Now that we have a good compile, we need to do a little project clean up. By project, I am referring to the group of files that comprise the entire design. The Hermes design has changed and evolved over time. Some functions were removed or superseded by new and improved ones. Other pieces of code were rewritten to be more efficient. The net result is that there are files included in the project that are unused. Since we don't need to update unused modules, it is best to remove them now. There are about two dozen unused files that you can remove. I have listed them in a text file that you can download.[9]

First, remove the files on the list from the project directory or subdirectory. If you are cautious, like I am, create a new directory outside of the *Quartus* project and move the files there. That way *Quartus* will not be able to find them, but if you make a mistake and remove a needed file, you can easily restore it. Next, in *Quartus*, under the **<Project><Add/Remove Files in Project>** menu, remove the files from the project. You might think that deleting (or moving) the file is sufficient, but *Quartus* keeps track of the files that it *knows* are in the project. You must remove these or *Quartus* will look for them (in vain, since you moved them) and not be happy about not finding them. After you remove all of the dunsel files, make sure to click **<apply>** and **<OK>**.[10]

Check the **Files** tab of the **Project Navigator** window to see a list of files in the project. See Figure 1. You should recompile the project to make sure that you did not accidentally remove something that is necessary. Before you do this, however, you need to remove the intermediate database files for past compiles. This will ensure that all traces of the files that you removed are gone from *Quartus* "memory" of compiles past. Go into the project directory and remove the two directories **db** and **incremental_db** along with their contents. Don't worry; *Quartus* will re-create them as soon as you run a compile, which you should now do. As before, *Quartus* should report some warnings, but no errors.

Update Wizard-Generated Modules

The Altera MegaWizard Plug-In Manager was used to generate some of the modules in the Hermes code. The wizard, as I call it, is software built into *Quartus* that helps you set parameters for Altera functions such as FIFO, RAM and ROM memories, phase-locked loops (PLLs) and other functions. The Hermes design uses four PLLs, four FIFO memories, three ROM memories, one RAM memory and one multiplier for a total of 13 Wizard generated modules. 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 II* version 15.0.

Let's now move our design to the Cyclone V family. With the design open, select **<Assignments><Device>** from the menu bar. Select **Cyclone V** in the **Family** field. A dialog box will appear asking if you want to remove all location assignments. This tells *Quartus* to remove the old pin assignments that will no longer be valid when we change to a different part. This is important, since the Cyclone III pin numbers have

Altera Part Numbers Explained, Sort Of

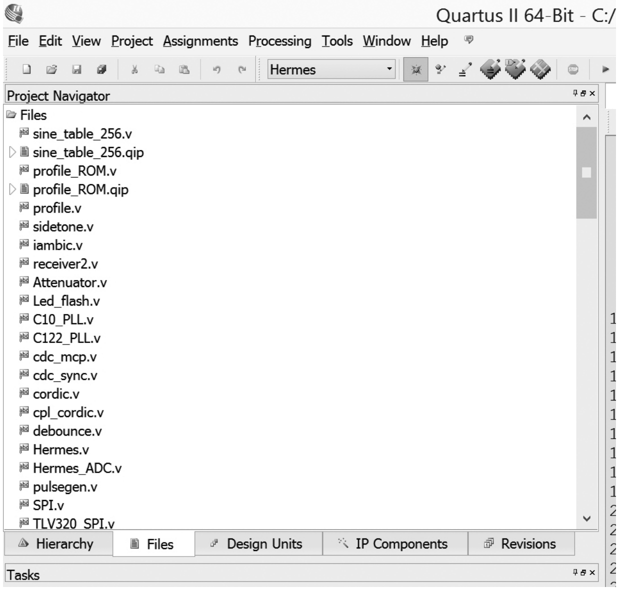
Just in case you are wondering what all those numbers mean in that long and involved FPGA part number, look no further. Our FPGA part number can be broken into 9 sections:

5C E F A9 F 23 C 8 N

The ***5C*** signifies that our part is in Altera's Cyclone V family of parts. Examples of other Altera part families are Stratix 5 (5S) and Arria 10 (10A). The ***E*** in our part number signifies Enhanced logic/memory, in other words, no embedded *hard-processor* or *high-speed transceivers* (the digital logic kind of transceiver, not the Amateur Radio variety). The ***F*** signifies that we have a *hard memory controller*, which is a DDR memory controller pre-built for us in silicon so we do not have to design one out of the FPGA fabric ourselves. The A9 tells us that this is the largest device in the family, with 301K Logic Elements (LEs). In contrast, the smallest member of the family, the A2, has only 25K LEs.

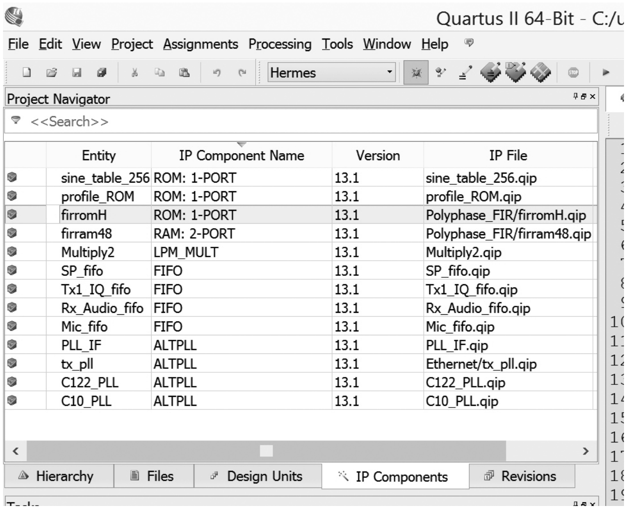
Moving along, ***F23*** represents the package type. F stands for Fine Line Ball Grid Array and 23 stands for the square package side dimension, 23mm. This package has 484 connections, each consisting of a solder ball on the bottom of the chip. The solder balls are arranged in a 22mm by 22mm square grid on 1mm centers. Don't try to mount this part with your American Beauty soldering iron![15]

The ***C*** stands for commercial temperature range (0°C to 85°C); there are two wider temperature ranges if needed. The ***8*** represents the speed grade. There are only three grades, 6 being the fastest (and most expensive). The 8 graded parts are the slowest (and cheapest), but still plenty fast enough for our application. As you would expect, grade 7 parts are in between 6 and 8 in performance. Last but not least, the N indicates lead-free packaging. No Ethyl for us, thank you.[16] More information than you ever wanted to know is available in the reference.[17]



Files
- sine_table_256.v
- sine_table_256.qip
- profile_ROM.v
- profile_ROM.qip
- profile.v
- sidetone.v
- iambic.v
- receiver2.v
- Attenuator.v
- Led_flash.v
- C10_PLL.v
- C122_PLL.v
- cdc_mcp.v
- cdc_sync.v
- cordic.v
- cpl_cordic.v
- debounce.v
- Hermes.v
- Hermes_ADC.v
- pulsegen.v
- SPI.v
- TLV320_SPI.v

Figure 1 — Project Navigator view of files in the project.



Entity	IP Component Name	Version	IP File
sine_table_256	ROM: 1-PORT	13.1	sine_table_256.qip
profile_ROM	ROM: 1-PORT	13.1	profile_ROM.qip
firromH	ROM: 1-PORT	13.1	Polyphase_FIR/firromH.qip
firram48	RAM: 2-PORT	13.1	Polyphase_FIR/firram48.qip
Multiply2	LPM_MULT	13.1	Multiply2.qip
SP_fifo	FIFO	13.1	SP_fifo.qip
Tx1_IQ_fifo	FIFO	13.1	Tx1_IQ_fifo.qip
Rx_Audio_fifo	FIFO	13.1	Rx_Audio_fifo.qip
Mic_fifo	FIFO	13.1	Mic_fifo.qip
PLL_IF	ALTPLL	13.1	PLL_IF.qip
tx_pll	ALTPLL	13.1	Ethernet/tx_pll.qip
C122_PLL	ALTPLL	13.1	C122_PLL.qip
C10_PLL	ALTPLL	13.1	C10_PLL.qip

Figure 2 — Project Navigator view of IP components in the project.

about the same chance of being the same as the Cyclone V pin numbers as my dog has of becoming President. (My cat agrees with me on this one.) This is especially true since the packages (QFP240 versus FBGA484) are completely different. So click **Yes** to remove them. To narrow your choices, select **FBGA** in the **Package** field, **484** in the **Pin count** field and **8** in the **Speed grade** field. Now select **5CEFA7F23C8** under Available Devices with a single click. Note that you will again have to confirm that you want to remove all location assignments, even though they have already been removed! Click **Yes** and then **OK**. That's it! You are now are using a Cyclone V part! Well, not the right part, 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.

To update the wizard-generated modules, we will use (are you ready for this?) the wizard itself! We will open each module in turn and tell the wizard to use the Cyclone V family and regenerate the module. This will work for all of the modules except the four PLLs and the ROM memory. We will handle them separately. To get started, click the **IP Components** tab of the **Project Navigator.** See Figure 2. You should see thirteen lines in the window. Leave the PLLs alone for now (**PLL_IF**, **tx_pll**, **C122_PLL** and **C10_PLL**) as well as the **firromH** module. Open **sine_table_256** by double clicking on it. The MegaWizard Plug-In Manager will start. In the upper right corner, the **Currently selected device family** will be **Cyclone III** and the **Match project/default** box will be checked. Uncheck this box and then select **Cyclone V** from the **Currently selected device family** drop-down menu. Click Finish twice and the wizard will update the module for you. Now repeat the same steps for the other 7 modules (profileROM, SP_fifo, firram48, Tx1_IQ_fifo, Rx_Audio_fifo, Multiply2 and Mic_fifo).

The firromH module must be handled differently, mainly because the designers broke one of the rules and modified the firromH.v file that the wizard created. There are reasons why they did this (which I am not going to elaborate on), but the consequences are that the new wizard-generated firromH.v file will over-write the modified old version. We will have to re-modify the new file with the old changes to make it work. Go ahead and open the firromH module in the wizard and convert it to the Cyclone V family just like you did with the other modules. After you do this, click on the **Mem Init** tab in the toolbar. We must specify an existing file name in order to satisfy the wizard, so click on the **Browse** button and select the **Polyphase_FIR** directory and pick any of the files that end in **.mif**. You will have to change the selection to **MIF files** in the drop-down **Files of type** box at the bottom of the window to make the .mif files visible. It does not matter which file you choose, since we are going to manually change it in the firromH.v file in the next step.

Now we are going to modify the firromH.v file that the wizard created for us. (Shh, don't tell the wizard!) In the *Quartus* Project Navigator pane, click on the **Files** tab, find the **firromH.v** file (hint: it is called "Polyphase_FIR/firromH.v" because it is in a project sub-directory) and open it by double-clicking on it. After line 43, add line 44:

parameter MifFile = "missing_file.mif";

Follow this with a blank line 45 to make things readable. Next go to line 88 and change it to read:

altsyncram_component.init_file = MifFile,

Make sure to type it exactly as shown, since capitalization and punctuation matter. Save it and close the file. We will delve more into why we made this change in the next column, when we dig into the code. Right now we need to finish up with the wizard by updating the PLL modules.

Unfortunately, Cyclone V PLLs are different from Cyclone III PLLs, so we cannot just upgrade them using the wizard. We must create new ones and replace the old ones with the new ones. First, remove the old PLL_IF, C10_PLL, C122_PLL and tx_pll files from the project using the Add/Remove Files in Project menu. Each of these modules will have several files (typically .v and .qip files); make sure that you remove all of them. Next remove the files from the project directory and sub-directories. (The **tx_pll** files are in the Ethernet sub-directory.) While you are at it, remove the **db** and **incremental-db** directories and their contents, just like you did before.

To create a new PLL module, open the wizard using **<Tools><MegaWizard Plug-In Manager>** and select **Create a new custom megafunction** from the list. From the list of functions, pick **Altera PLL v13.1** from the PLL submenu. In the output file box append the name (for example, **PLL_IF_new**) after the string that represents the project directory. This will name your module and place it in the project directory. All four

Table 1
Wizard Settings for New PLL Modules

	PLL_IF_new	*tx_pll_new*	*C10_PLL_new*	*C122_PLL_new*
Reference Clock	122.88 MHz	50.0 MHz	10.0 MHz	122.88 MHz
Number of clocks	4	4	2	2
Multiply Factor (M)	4	10	64	9
Divide Factor (N)	1	1	2	3
outclk0 cascade?	N	N	Y	Y
outclk0 Divide Factor (C)	40	4	32	192
outclk0 output	12.288 MHz	125 MHz	n/a	n/a
outclk0 phase shift	0°	0°	0°	0°
outclk1 cascade?	N	N	N	N
outclk1 Divide Factor (C)	160	4	125	24
outclk1 output	3.072 MHz	125 MHz	80 kHz	80 kHz
outclk1 phase shift	0°	90°	0°	0°
outclk2 cascade?	Y	N	n/a	n/a
outclk2 Divide Factor (C)	256	40	n/a	n/a
outclk2 output	n/a	12.5 MHz	n/a	n/a
outclk3 cascade?	N	N	n/a	n/a
outclk3 Divide Factor (C)	40	200	n/a	n/a
outclk3 output	48 kHz	2.5 MHz	n/a	n/a

PLL modules have these common settings:

- Device Speed Grade: 8
- PLL Mode: Integer-N PLL
- Operation Mode: direct
- Enable locked output port: checked
- Enable physical output clock parameters: checked

Set the other parameters for each module to what I have listed in Table 1. Leave all other parameters set to their default settings. When you click **Finish** and **Exit** after specifying all the parameters, *Quartus* will ask you if you want to add the new IP to the project. Click **Yes**. Since the PLL modules need to be added to the project eventually, this will save you a step later.

The last step is to open the source file that instantiates each PLL, update the module name and check (and correct, if necessary) the module connections. The **tx_pll** is used in the **rgmii_send.v** file in the Ethernet subdirectory. The other three are instantiated in the top level **Hermes.v** file. I will guide you through the first one, and you can follow the same procedure on the other three on your own. (You didn't think I was going to do *all* of it for you, did you?) Open the top level **Hermes.v** file and also the **C122_PLL_new.v** file. Go to line 1385 in **Hermes.v** and you will see the instantiation of C122_PLL. The instantiated name is PLL_inst, and the port names are inclk0, c0 and locked. Observe that port **inclk0** is connected to **_122MHz**, port **c0** is connected to **osc80khz** and port **locked** is not connected to anything. Now look at the **C122_PLL_new.v** file. You will see that there are now four ports instead of three: **inclk0** is now called **refclk**, **c0** is now called **outclk_1** and **locked** remains unchanged. The new input is called **rst;** we will not use it. Change line 1385 to read:

```
C122_PLL_new PLL_inst (.refclk(_122MHz),
.outclk_1(osc_80khz), .locked( ), .rst( ) );
```

The C10_PLL_new and PLL_IF_new modules will require similar changes.

The astute reader will notice that the wizard allows you to turn off the **locked** output when it is unused, but the new PLLs all have an **rst** input that cannot be disabled. Ideally this input should be connected to reset logic; however, we will save code improvements for a later time. I need to call your attention to one other change that I slipped in while you were not looking. I changed the reference clock of the **tx_pll** module from 125 MHz to 50 MHz. I did this out of necessity, since the CVA9 does not have a 125 MHz clock input! It does have a 50 MHz clock input, but since we have not added it to the top-level **Hermes.v** source file yet, just leave it at 125 MHz. We will fix it shortly.

Now that you are experienced in matching up old port names to new port names, this would be a good time to go back and check the other nine wizard-generated modules that we updated to make sure that the port definitions in each module's .v file match up with the ports called out at the module's instantiation. Here's a quick hint: all of them are instantiated in **Hermes.v** except for sine_table_256 (sidetone.v), Multiply2 (sidetone.v), profile_ROM (profile.v) firromH (Polyphase_FIR/firx2r2.v) and firram48 (Polyphase_FIR/firx2r2.v).

As a short aside, I keep a note pad handy to write down things like "change 125M clock to 50M" as a note to myself. When you are updating the code, you will likely perform many tasks out of order and it is easy to forget a simple change that you queued up in your memory and then forgot about it.

At this point, we are finished with *Quartus II* version 13.1. Close *Quartus* and re-open the project in *Quartus II* version 15.0. 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 15.0 with the correct FPGA part number. The light at the end of the tunnel is coming into view, and it is not an oncoming train!

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. We must account for *every one* of the Hermes signals, whether it is to remove it, change it to match the new CVA9 hardware or just connect it to its counterpart in the new design. We must also account for *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. In order to be able to do all of this cross checking, we need to map the old pin names (in this case from the Hermes board) to our new pins on the BeMicroCVA9 board. Some of these signals connect to parts on the BeMicroCVA9, and some connect directly to the HF2 and TX2 boards that are plugged into the BeMicroCVA9.

What we need is a table that shows the old name alongside the new name and the new FPGA pin number. (We will use the pin numbers in the next section.) You could figure this out for yourself, but I have created a file for you containing a table of all of the signal names in the design to give you a head start. This **Hermes_6_to_IQ2_pins** table will tell us which pins map directly onto new pins and which do not.[11] An excerpt of this table (showing only the signals that we need to change) is shown in Table 2. All of the changes will be made to the top level **Hermes.v** file.

A quick look at the full table will reveal that most Hermes signals have equivalent (although differently named) BeMicroCVA9 signals. We can leave these alone. The other signals fall into three categories:

1) The Hermes signal has a different or shared function than the CVA9 signal.

2) The Hermes signal does not exist in the CVA9 design.

3) The CVA9 signal does not exist in the Hermes design.

In the first case, we must modify the

Hermes code to connect to the CVA9 hardware that is different from the Hermes hardware. As an alternative, we can choose to not implement the Hermes function on the different CVA9 hardware. This involves removing (typically by commenting out) code that connects to the removed pins. As we will explain next, you have to be careful when removing inputs.

In the second case, we can simply remove Hermes code that does not have CVA9 hardware associated with it. We must be careful to follow Hermes inputs all the way to their destinations and remove them cleanly. We do not want any floating inputs. There may be one or more required inputs to the Hermes code that came from hardware that does not exist on the CVA9. We will have to add new code to create these signals and set them to a valid state.

In the third case we must add code to the Hermes design to connect to the CVA9 hardware that does not exist in the Hermes design. As an alternative, we can choose to ignore the new hardware, but we must still drive any output pins to some known state to avoid hardware problems later.

Here are the index numbers (from the **Hermes_6_to_IQ2_pins** table) that belong to each category:

Category 1: 4, 5, 50, 51, 52, 53

Category 2: 28, 49, 67, 70, 73-76, 78-85, 91-104, 113-115

Category 3: 21, 22, 116-120

Category 1 Changes

These 7 pins all revolve around a hardware difference between the Hermes and the CVA9/HF2/TX2 hardware. Hermes has a 31 dB step RF attenuator and a single audio CODEC for receive audio output and microphone audio input. These two devices have separate serial interfaces (3-wire for the attenuator and 3-wire for the CODEC). The HF2 receiver has the same attenuator and CODEC (which is used for receive audio output only), but they share clock and data lines, each having a separate chip-select. This makes the interface 4 lines to both parts. To complicate things, the TX2 transmitter has another CODEC (used only for microphone audio input) that shares the same clock and data lines, but with its own separate chip select. So now the new five-line interface must communicate with three parts over common clock and data lines using three different chip-selects. Rather than devise logic to adapt the two Hermes ports to the special five-line CVA9/HF2/TX2 interface, I have opted to just disable the HF2 and TX2 CODECs by tying their chip-selects to the inactive state. The PowerSDR^{TM} software can use the sound card in place of the CODECs, so this does not create a hardship. We can go back later and add the code in if we want to.

Table 2
Excerpt of the Hermes_6_Sept_to_IQ2_Pins File

Index	*Hermes name*	*Hermes FPGA pin*	*HF2 Name*	*TX2 Name*	*CVA9 J2 pin*	*CVA9 name*	*CVA9 FPGA pin*	*Description*
4	ATTN_DATA	39	SPI_DATA	SPI_DATA	62*	EG_P58	V20	Data Output To Attenuator
5	ATTN_CLK	22	SPI_CLK	SPI_CLK	64*	EG_P59	U20	Clock Output To Attenuator
21	INA14		INA14	-	41	EG_P17	AB20	Input Data From ADC
22	INA15		INA15	-	43	EG_P18	Y20	Input Data From ADC
28	SHDN	194	-	-	-	-	-	Shutdown Output to ADC
49	CMODE	230	-	-	-	-	-	Mode Select Output To CODEC (I2C or SPI)
50	nCS	231	PH_CODEC_nCS	-	60	EG_P57	V19	Chip Select Output To CODEC
51	nCS	231	-	MIC_CODEC_nCS	57	EG_P24	Y21	
52	MOSI	226	SPI_DATA	SPI_DATA	62*	EG_P58	V20	SPI Data Output To CODEC
53	SSCK	224	SPI_CLK	SPI_CLK	64*	EG_P59	U20	SPI Clock Output to CODEC
67	PHY_CLK125	149	-	-	-	-		125 MHz Clock Input From PHY PLL
70	CLK_25MHZ	33	-	-	-	-	-	25 MHz Clock Input From PHY oscillator
73	SCK	68	-	-	-	-	-	Clock Output To MAC EEPROM
74	SI	38	-	-	-	-	-	Data Output To MAC EEPROM SI pin
75	SO	70	-	-	-	-	-	Data Input From MAC EEPROM SO pin
76	CS	87	-	-	-	-	-	Chip Select Output To MAC EEPROM
77	NCONFIG	63	-	-	-	RECONF	G6	Reload FPGA From Config Prom When High
116	-	-	DRV_CLK_OUT_N	-	67	EG_P29	U21	Output To HF2: Drive 122.88 MHz to J1 pin 5
117	-	-	-	DAC_CLK	3	RESET_EXPn	U13	Output To TX2: Clock To DAC
118	-	-	-	EN_RX_ANT	12	EG_P37	M7	Output To TX2: T/R Switch
119	-	-	-	-	-	DDR3_CLK_50MHZ	H13	Input From 50 MHz Oscillator
120	-	-	-	-	-	CLK_24MHZ	M9	Input From 24MHZ Oscillator

*shared pins

As they used to say in college, it is left as an exercise for the student.

We will leave the signals at index 4 (**ATTN_DATA**) and index 5 (**ATTN_CLK**) alone, which will allow normal control of the RF attenuator. We will remove the signals at index 50 and 51 (**nCS**), index 52 (**MOSI**) and index 53 (**SSCK**). Open **Hermes.v** and look at lines 154 to 156. Rather than delete the lines of code that we might want to add back in someday, just comment them out by adding two slashes (//) at the beginning of each line. To complete this change, we must also remove the signals that drove these outputs. Go to line 519 and delete the signals **nCS**, **MOSI** and **SSCK** from inside the parentheses. (Yes, this will leave an empty field between the parentheses. This is how you specify no connection.) This effectively disconnects the **.nCS**, **.MOSI** and **.SSCK** ports of the TLV320_SPI module from the top level outputs that no longer exist. While this is not a complete removal of the TLV320_SPI module, it is close enough; *Quartus* will remove the unused logic for us. We will still have the ability to easily connect it back up at a future date when we are ambitious enough to combine its outputs with the attenuator interface and make the CODECs work again.

Now that we have removed the signals for the Hermes CODEC, we must define and drive the two new CODEC chip selects to their inactive state. Since they are active-low, we will drive them high. Add these two lines right after the SSCK port definition that you just commented out:

```
output PH_CODEC_nCS,
output MIC_CODEC_nCS,
```

Insert the following lines of code in a convenient place. Right after the module definition around line 237 is a good place:

```
assign PH_CODEC_nCS = 1'b1;
assign MIC_CODEC_nCS = 1'b1;
```

Category 2 Changes

These are perhaps the easiest changes to make. Inputs are handled differently than outputs. Outputs are handled as above: comment out the output pin and remove (or comment out) the source of the signal. Simply search for each signal name in turn and comment out its definition and its source. Inputs must be tied to a known (typically inactive) state after the input pin definition is commented out. We must also define an internal pin to replace the input pin definition that we commented out. First, let's identify the inputs from the list of Category 2 changes listed above. They are index numbers 67, 70, 75, 80, 91, 92, and 94 to 97. Find each of them in the full table, locate the corresponding input pin definitions in **Hermes.v** and comment them out. Note that the inputs CLK_25MHZ, ANT_TUNE, IO2, IO4, IO5, IO6 and IO8 are unused in the code, so they require no further changes. The signals SI and ADCMISO do need to be set to a known state. To do this, insert the following lines of code in a convenient place. Right after the Category 1 lines you added above is a good place:

```
wire SO;
assign SO = 1'b0;
wire ADCMISO;
assign ADCMISO = 1'b0;
```

The last input we need to handle is special: the PHY_CLK125 clock input. Remember from our scratchpad memo notes that this clock does not exist in the CVA9. We have already changed the tx_pll module to use a 50 MHz clock, which we will now add and connect up in place of the missing 125 MHz clock. Add the following code after line 166 (just below the **input PHY_CLK125** line that you commented out:

```
input DDR3_CLK_50MHZ,
```

Now search for PHY_CLK125 (ctrl-F opens a find window in *Quartus*) and change it to **DDR3_CLK_50MHZ** in two places: within the parentheses in the network module instantiation (around line 400) and near the end of the file in the always block heartbeat LED definition. Yes, it will make the heartbeat LED flash a bit slower, but that is acceptable.

To remove the outputs, comment out each output line in the module pin definitions at the beginning of the file (just like you did with the Category 1 outputs). In addition, remove the pin from inside the parentheses in a module instantiation (again, just like you did with the category 1 outputs) or comment out the assignment statement in which it appears. The signals **USEROUT0** – **USEROUT7** are a special case because they are assigned to the signals **Open_Collector[1]** – **Open_Collector[7]**, respectively. You must comment out the assignment (within the parentheses) of **Open_Collector** in the instantiation of the High_Priority_CC module (around line 1200) as well as the following assignment (around line 1144):

```
wire [7:0] Open_Collector;
```

Category 3 Changes

The index 21 and 22 changes widen the ADC data bus from the Hermes code 14 bit width to the HF2 receiver ADC width of 16 bits. We need to change the code in the **always** block that defines the variable **temp_ADC**. This variable is already 16 bits wide, but only the top 14 bits are connected to the 14 **INA** inputs from the ADC. Change line 135 to read:

```
input [15:0] INA,
```

Search for temp_ADC (around line 870) and change the **always** block to read as follows:

```
always @ (posedge C122_clk) begin
  temp_DACD       <= {DACD, 2'b00};
  if (RAND) begin
    if (INA[0])
      temp_ADC <= {~INA[15:1], INA[0]};
    else
      temp_ADC <= INA;
  end
  else
      temp_ADC <= INA;
end
```

For indexes 116 and 118 we need to add two new output signals and assign values to them. Index 117 is the output clock to the transmit DAC. On the Hermes board, the 122.88 MHz oscillator feeds the DAC directly without FPGA involvement. The TX2 transmitter DAC requires a clock from the FPGA, so we must add it. Finally, index 120 is an unused 24 MHZ oscillator input. Even though it is not currently used, we need to assign it to an input so we can fix the input pin location in the pin list. Add the following pin definitions to the Hermes module pin list at the beginning of the **Hermes.v** file:

```
output DRV_CLK_OUT_N,
output DAC_CLK,
output EN_RX_ANT,
input CLK_24MHZ,
```

If you insert these at the end of the pin list, remember that a comma separates each pin definition, and there is no comma after the last one. Now insert the following code in a convenient place (after your previous Category 2 code additions is a good place):

```
assign DRV_CLK_OUT_N = 1'b1;
assign DAC_CLK = _122MHz;
assign EN_RX_ANT = 1'b1;
```

Go ahead and compile again, just to make sure that you didn't forget a semicolon or make some other easy-to-fix syntax error.

Add New Location Properties

Do you remember those location properties that we removed when we changed part numbers? We now have to add them back into the design, except that we want to add the pin numbers for our new device in place of the old numbers that we removed. This is why I included the pin numbers in Table 2.

The best way to add new location properties to the design is to write a script file that contains a line for each new location assignment. The format for each line is:

set_location_assignment PIN_*xxxx* –to *signal_name*

where:

xxxx is the device pin number, and

signal_name is the pin name from the top design file (**Hermes.v**).

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.[12]

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 (or maybe you just don't believe me) 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 another compile to make sure things are as they should be.

Update SDC Timing Constraints

The last task we must undertake is to review 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. I will cover this in my next column. In the meantime, take a look at the file to become familiar with it. Did I just give you a homework assignment? Sorry.

Compile-Debug-Repeat

The focus of our efforts has been on obtaining a good compile of our code under the new *Quartus* version while targeting the new FPGA part. That said, a successful compile does not necessarily mean we have a working design. Now is the time to review all of those *Quartus* warnings that we have so blithely been ignoring all this time. Most (if not all) of them can be ignored, but we must make sure of this. The cause of the ones that cannot be ignored must be fixed. The final step, of course, is to load the compiled programming file into the BeMicroCVA9 and test it to make sure that it works. I will cover review of warnings and how to fix them, timing constraints update and 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]

Table 3
Files

Quartus archive of original source code:	Hermes_6_Sept.qar
Table of pin cross references:	Hermes_6_Sept_to_IQ2_pins.pdf
List of unused files:	Hermes_6_Sept_unused_files.txt
Tcl script to reassign location properties:	Hermes_6_Sept_map_pins.tcl
Quartus archive of ported source code:	Hermes_6_Sept_ported.qar

What's Next?

Now that you know how to port FPGA code to new devices, what can you do with this skill? The openHPSDR project is open source, and the Apache Labs Anan series of transceivers are all powered by open-source FPGA firmware. The FPGA code to implement any or all of the features of these transceivers is available for your use. When a new feature comes out, you can look at how it is done and integrate that function into your radio. Better yet, you can add your own feature and show everyone else how to improve their own rigs. That is the true benefit of open-source!

Each openHPSDR board has an on-board FPGA and Verilog code to match. All of it is available from the openHPSDR repository, and you are now qualified to port it to any new hardware that you can scrounge up.[14] The tools that you have used today are the very same tools that the developers use when they write or update the code.

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] The BeMicroCVA9 board is available from from Arrow Electronics: **parts.arrow.com/item/detail/arrow-development-tools/bemicrocva9**.

[3] The UDPSDR-HF2 receiver board is available from Arrow Electronics: **parts.arrow.com/item/detail/arrow-development-tools/udpsdr-hf2.**

[4] The UDPSDR-TX2 transmitter board is available from Arrow Electronics: **parts.arrow.com/item/detail/arrow-development-tools/udpsdr-tx2**.

[5] The UDPSDR-HF1 receiver board is available from Arrow Electronics: **parts.arrow.com/item/detail/arrow-development-tools/udpsdr-hf1**.

[6] For more information about Hermes-Lite see: **github.com/softerhardware/Hermes-Lite/wiki**.

[7] To download the free Altera Web Edition software, go to: **dl.altera.com/?edition=web**.

[8] 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_6_Sept.qar>**. It is also available for download from the ARRL *QEX* files web page. Go to **www.arrl.org/qexfiles** and look for the file **1x16_Cowling_Hands_On_SDR.zip**.

[9] The list of unused files is available from the SDRstick SVN in the same directory as listed in Note 8. The file name is **<Hermes_6_Sept_unused_files.txt>**. This file is also part of the **1x16_Cowling_Hands_On_SDR.zip** file, also as listed in Note 8.

[10] dunsel, noun, (slang, from Star Trek) a part that serves no useful purpose.

[11] The cross reference of Hermes to IQ2 pins is available from the SDRstick SVN in the same directory as given in Note 8. The file name is **<Hermes_6_Sept_to_IQ2_pins.xls>**. The file is also included in the **1x16_Cowling_Hands_On_SDR.zip** as given in Note 8.

[12] The pin location Tcl script file is available from the SDRstick SVN in the same directory as given in Note 8. The file name is **<Hermes_6_Sept_map_pins.tcl>**. The file is also included in the **1x16_Cowling_Hands_On_SDR.zip** file.

[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_6_Sept_ported.qar>**. This file is also included in the ZIP file, as listed in Note 8.

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

[15] American Beauty soldering irons of old were massive and the larger ones could solder copper pipes. Like the term "boat anchor," the term is an affectionate name for a tool of the past. Lo and behold, they are still in business! I especially like the handheld unit shown at **americanbeautytools.com/Soldering-Irons/19/features**.

[16] An obscure and wholly unwarranted reference to tetraethyllead $(CH_3CH_2)_4Pb$, an octane booster added to gasoline from about 1920 until the early 1990s in the USA. Premium gasoline was referred to as "Ethyl" to us old timers.

[17] More information on part numbers can be found in Altera's Cyclone V Device Overview at **altera.com/en_US/pdfs/literature/hb/cyclone-v/cv_51001.pdf**.