# Introduction to MAX 10 ADCs Finger Temperature Tutorial

## For the BeMicro MAX 10 FPGA Evaluation Kit

Version 14.0 10/9/2014 Tutorial

## 1. GETTING STARTED AND INSTALLATION OF TOOLS

BeMicro Max 10 is a FPGA evaluation kit that is designed to get you started with using an FPGA. BeMicro Max 10 adopts Altera's non-volatile **MAX® 10 FPGA** built on 55-nm flash process.

MAX 10 FPGAs revolutionize non-volatile integration by delivering advanced processing capabilities in a low-cost, instant-on, small form factor programmable logic device. The devices also include full-featured FPGA capabilities such as digital signal processing, analog functionality, Nios II embedded processor support and memory controllers.

The BeMicro Max 10 includes a variety of peripherals connected to the FPGA device, such as 8MB SDRAM, accelerometer, digital-to-analog converter (DAC), temperature sensor, thermal resistor, photo resistor, LEDs, pushbuttons and several different options for expansion connectivity.

**Before continuing with this Tutorial, ensure that the Altera tools and drivers have been installed. Please refer to the *BeMicro MAX 10 Getting Started User Guide* for instructions.**

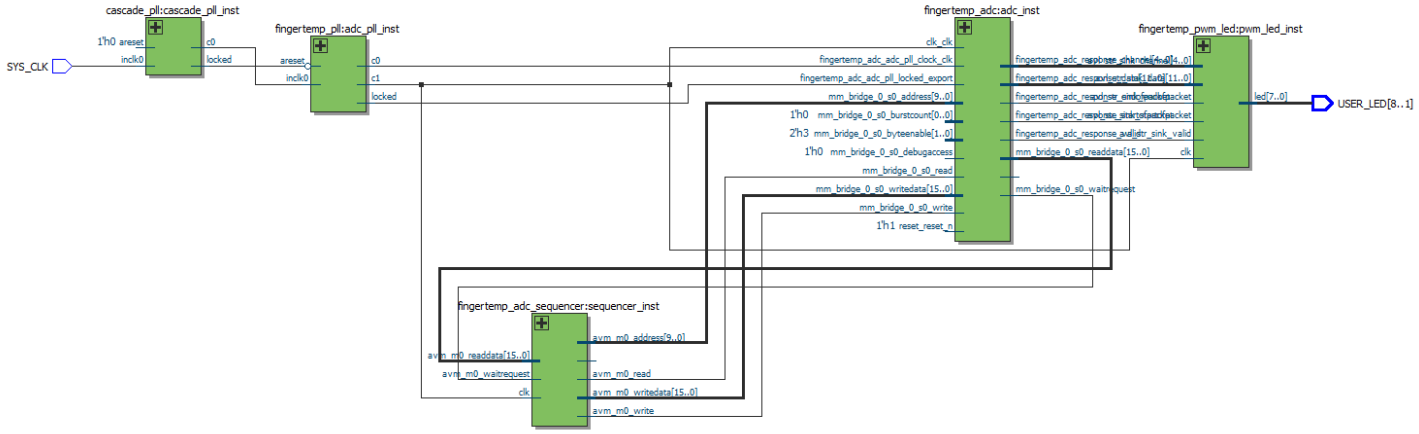
## 2. FINGER TEMPERATURE EXERCISE

### 2.1 Overview

The purpose of this exercise is to become familiar with the Max10's ADC and get familiar with using IP functions as well as integrating them into an HDL (Hardware Description Language) environment, in this case VHDL.

The final design will use individual PWMs for the eight user-LEDs on the board to show the relative temperature from room temperature to finger temperature like a color bar where the "highest" LED goes from low to high intensity before lighting up the next LED in line.

The QuartusII RTL Viewer produces the following picture of the completed HDL design and is shown here to give you an idea of what the HDL in this lab example describes.



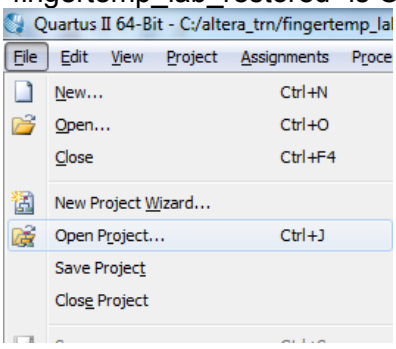

(Note: There are two PLLs in this design because the MAX 10 FPGA's ADC requires a specific PLL as its clock source. It is called fingertemp_pll in the diagram above. The BeMicroMax10 board's clock source can only drive directly into the other PLL. We called it cascade_pll in this case.)

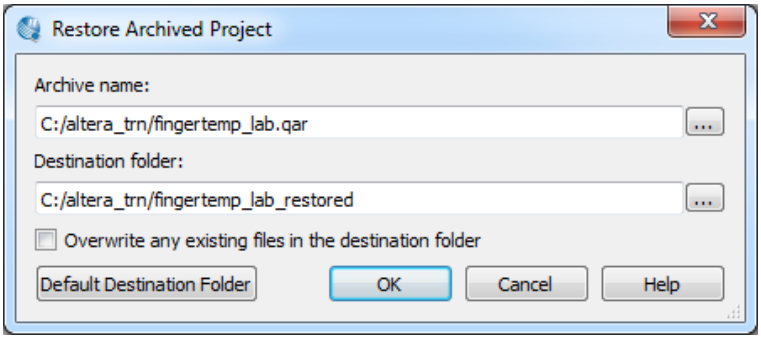
Modules fingertemp_pll and fingertemp_adc have been deleted from the original design and it is your task to create and insert them before compiling the design. It is important that you use the exact names and set the correct parameters according to the instructions below for them to fit and operate correctly with the top level design file.

## 2.2 Lab Instructions

Copy the fingertemp_lab.qar to a suitable placement, e.g. c:\altera_trn\ .

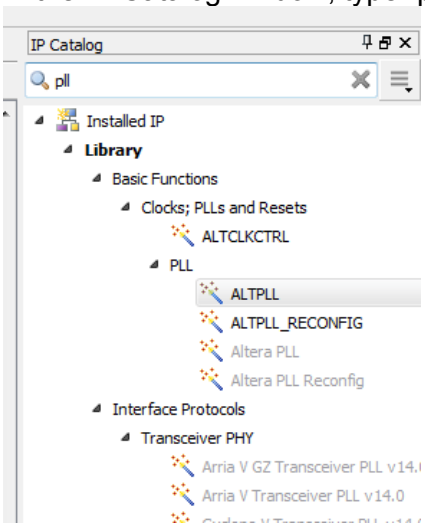
**Open the project:**
The project is stored in the format Quartus Archive ".qar".
From the File menu choose Open Project and select fingertemp_lab.qar. Accepting the placement suggestion "fingertemp_lab_restored" is OK for this exercise.



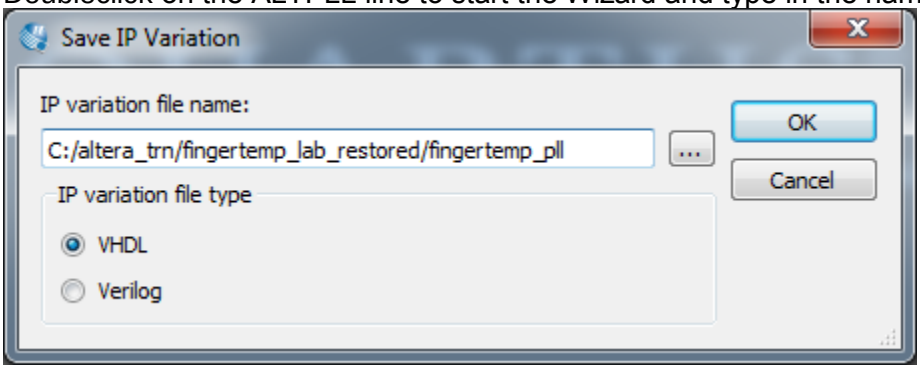




This project already contains the correct device settings and pin assignments that fits the BeMicro Max10.
If you try to compile at this point, you will get shouted at because critical design elements are missing…

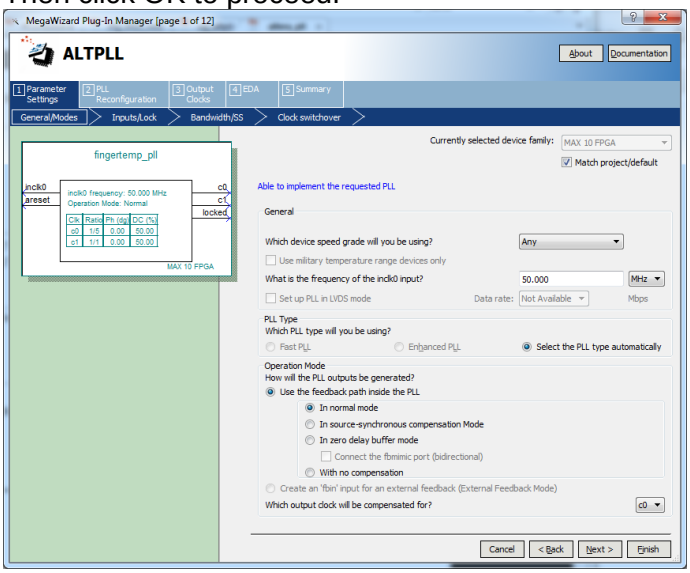
**Create the fingertemp_pll module:**
In the IP Catalog window, type “pll” to filter for relevant functions in the library:




Doubleclick on the ALTPLL line to start the Wizard and type in the name “fingertemp_pll”.

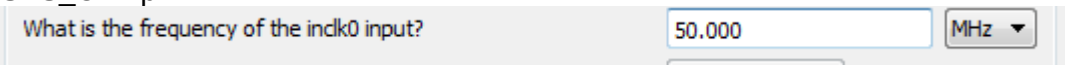



Then click OK to proceed.

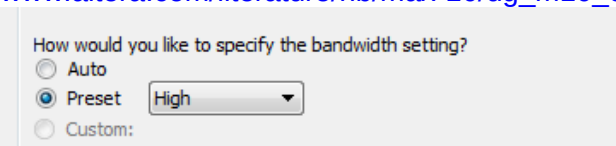



Under “1: Parameter Settings” tab change:

On the “General/Modes” tab; change the inclk0 input to 50MHz which corresponds to our source on the SYS_CLK pin.

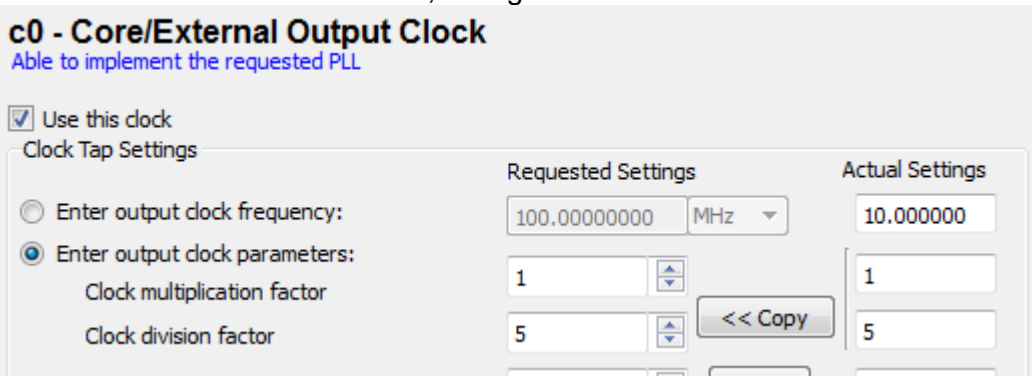



On the “Bandwidth/SS” tab; change the Bandwidth Setting to Preset High. This is to maintain low jitter for our ADC despite cascading the clock through two plls(more info on cascading PLLs in www.altera.com/literature/hb/max-10/ug_m10_clkpll.pdf ) :

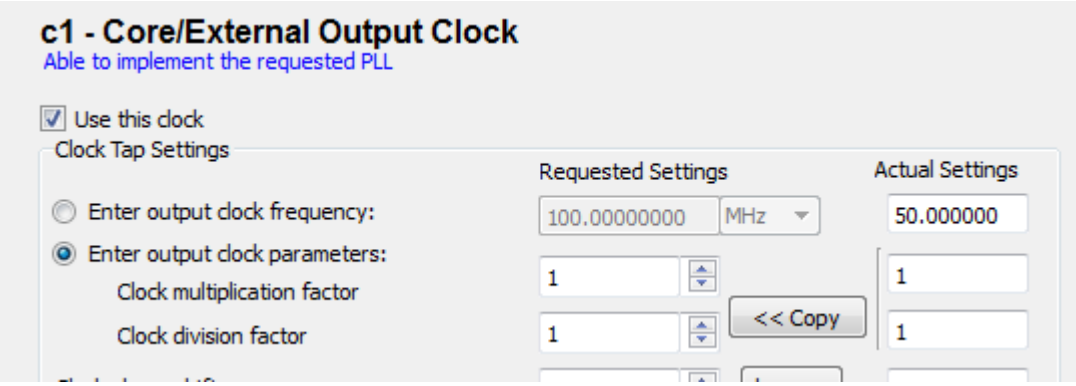



Under “3: Output Clocks” tab change:

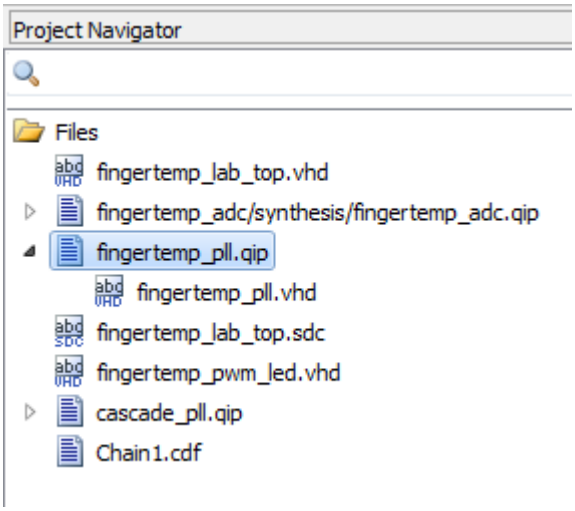
The ADC requires a 10MHz clock from the c0 output of PLL_1 in the 10M08 device on our BeMicro Max10 board. On the “clk c0” tab; change the division factor to “5”




On the “clk c1” tab; tick the “Use this clock” checkbox and leave the rest of the parameters at their defaults:



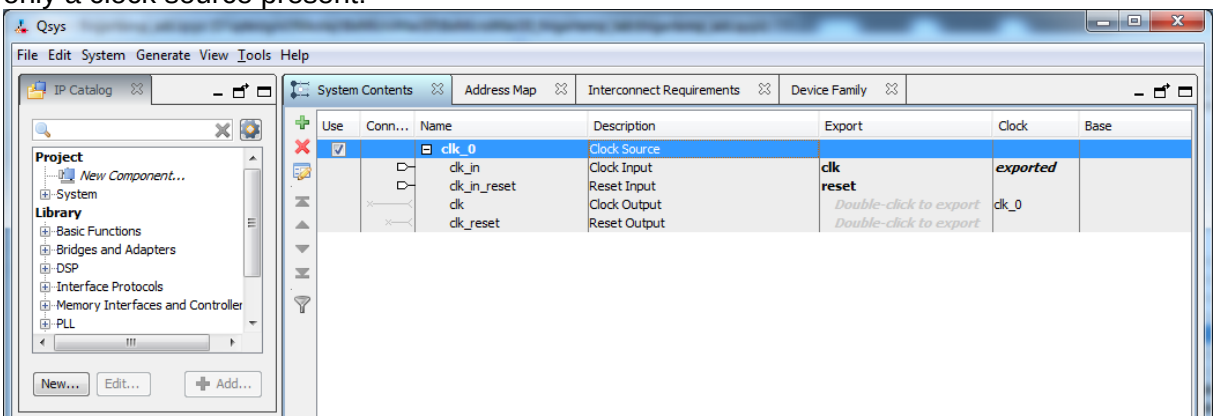
Click the “Finish” button and note that the resulting files were added to the Files tab of the Project Navigator:



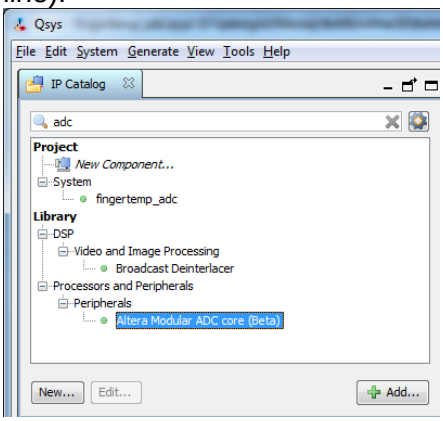

(Note: A *.qip file lists all library files necessary to synthesize the desired functionality specified in the wizard)

**Create the fingertemp_adc subsystem:**
Now we will create the ADC module with the desired parameters:
(Note: The ADC is likely often intended to be used in a system with either a processor or another “intelligent” master. The parameterization of this module therefore belongs in the more advanced system builder tool called Qsys. In our exercise we will however use the ADC in a very basic way…)

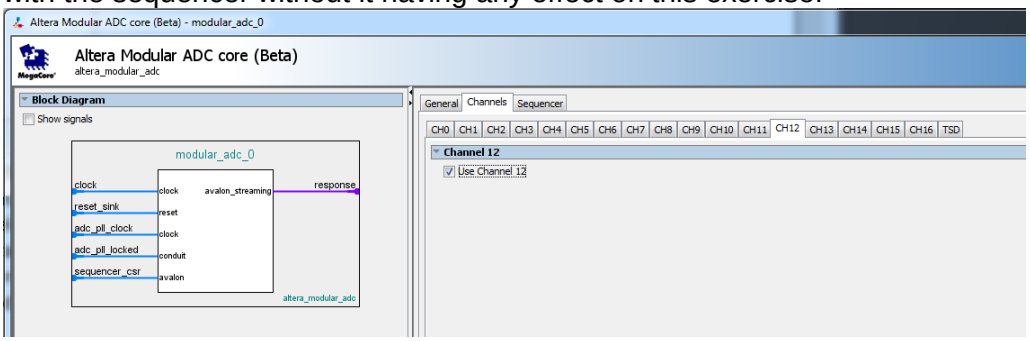
Open Qsys either from the Tools menu or click its symbol  on the toolbar.
If you are prompted with an Open File window, please press Cancel. You should now have a Qsys GUI with only a clock source present:



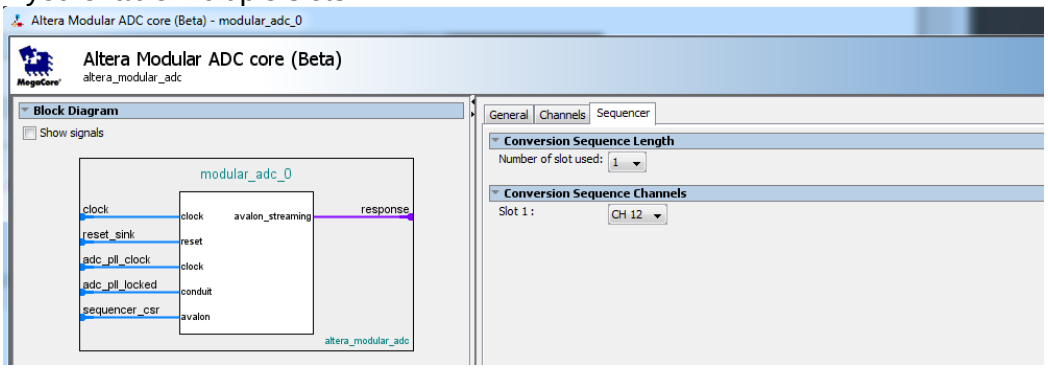
In the IP Catalog; search for ADC, select the “Altera Modular ADC core (beta)” and click Add (or doubleclick its line).



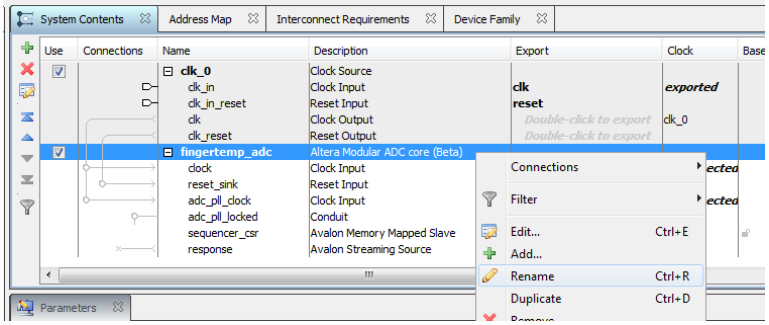

A new Wizard window will appear and we can set our desired parameters. In this case we want it to constantly spit out its samples, but we still want to check and control it’s status and command registers, so we should select the Core Variant to be “Standard sequencer with **external** sample storage” on the General tab.
For our exercise we are only interested in channel 12 which is connected to the thermal sensor on our BeMicro Max10 board. (You could also experiment with channel 4 which is connected the photo resistor…)
In our case, the fingertemp_pwm_led.vhd module is designed to only look for samples from channel 12. All other channels are disregarded. In other words you can enable any channels in addition to 12 as well as play with the sequencer without it having any effect on this exercise.



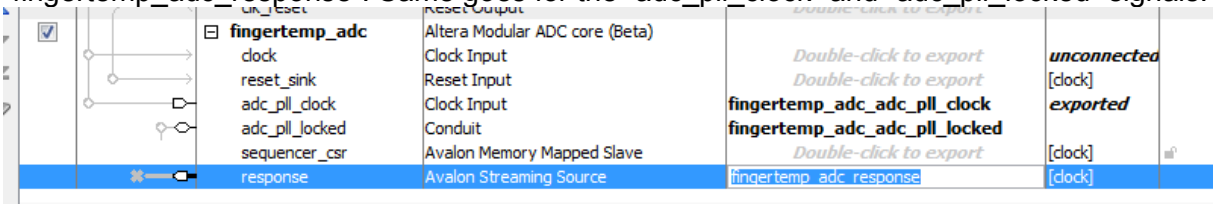

The sequencer settings defaults to no channel, so make sure you select channel 12 in at least one of the slots if you enable multiple slots.



Back in Qsys GUI, rename the recently added “Altera Modular ADC core” to “fingertemp_adc” by selecting the line and pressing F2 (as in Excel) or right click and chose Rename.

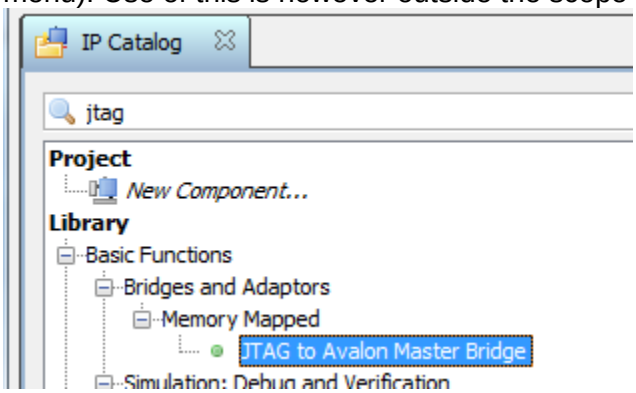



The ADC’s samples are streamed out of the Avalon Streaming Source interface called “response”. We want to connect this to our top level design file and therefore want to export it out of this module. Doubleclick in the export column on this line and click enter with the default name that should occur as “fingertemp_adc_response”. Same goes for the “adc_pll_clock” and “adc_pll_locked” signals:

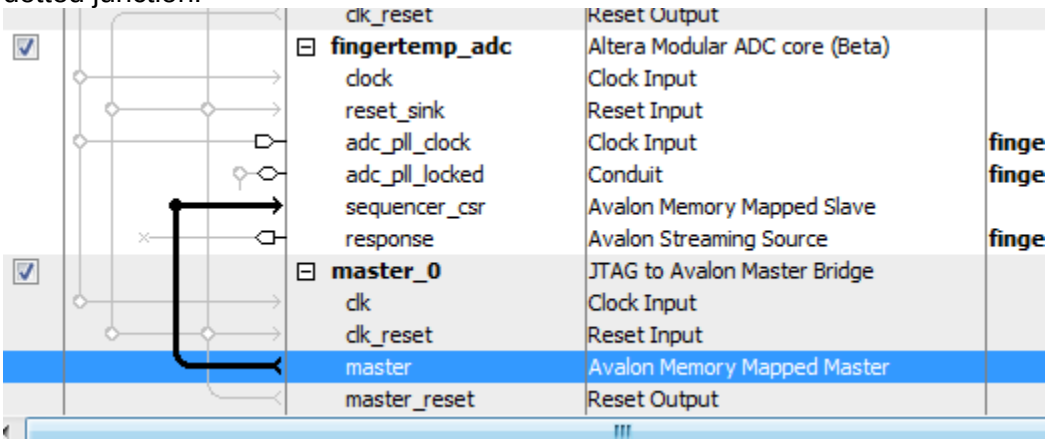



The ADC in “Standard Sequencer with external sample storage” mode also has a control/status memory mapped slave interface called “sequencer_csr”. We will not use it in this exercise, but we still want to connect it for potential design expands as well as debug trough JTAG port.

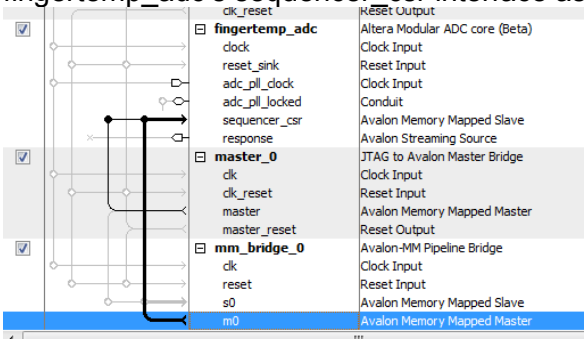
In Qsys IP catalog, search for “jtag” and add the “JTAG to Avalon Master Bridge”. This gives you a master interface to talk through with the “System Console” tool which is a part of the QuartusII installation(Tools menu). Use of this is however outside the scope of this exercise.

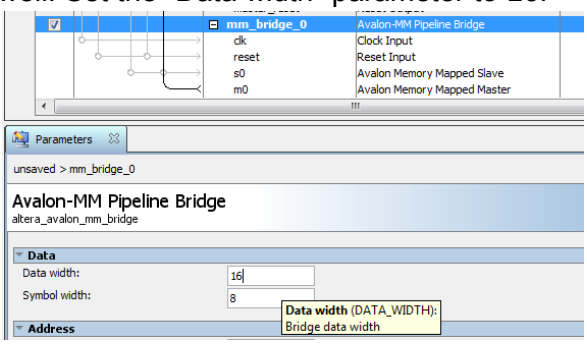


Connect the master_0's master interface to the fingertemp_adc's sequencer_csr interface by clicking on the dotted junction.

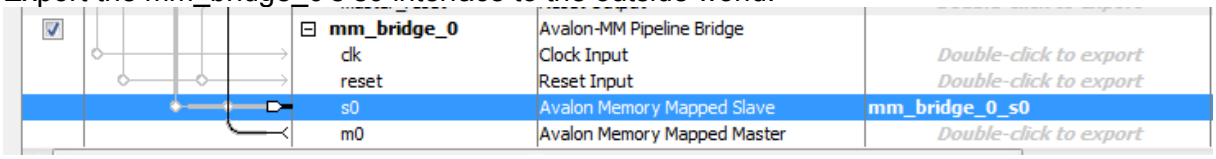



Add a "Avalon-MM Pipeline Bridge" from the IP catalog to our Qsys system, and connect it its master to the fingertemp_adc's sequencer_csr interface as well. Set the "Data width" parameter to 16.

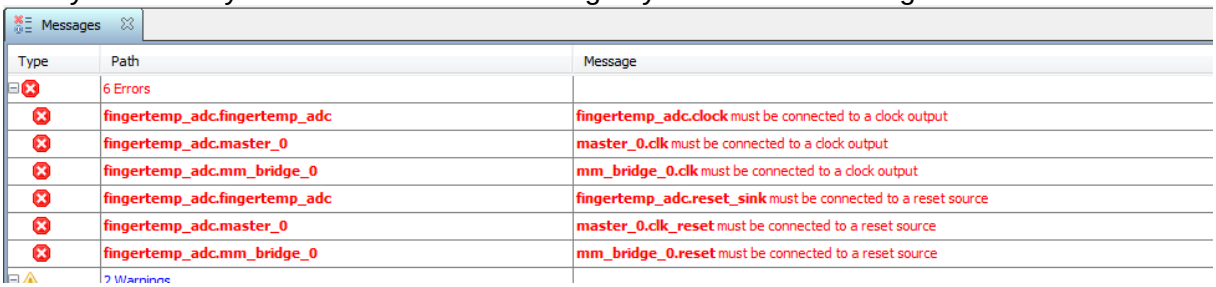


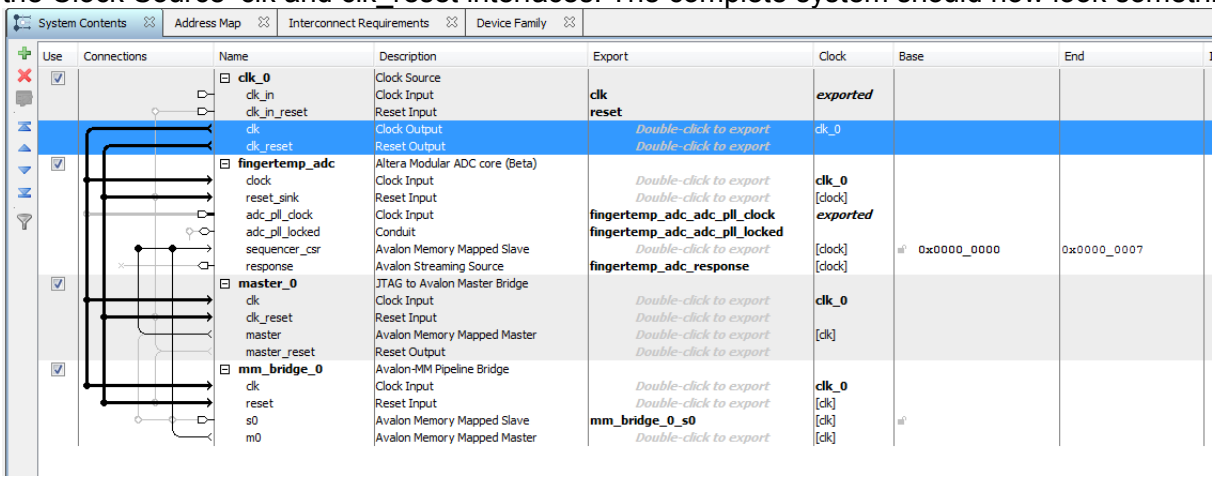

Export the mm_bridge_0's s0 interface to the outside world:




Now you will likely have 6 errors in **red** staring at you from the Messages window similar to this:



Obviously we need to connect our components' clock and reset interfaces to something, which in this case is the Clock Source' clk and clk_reset interfaces. The complete system should now look something like this:

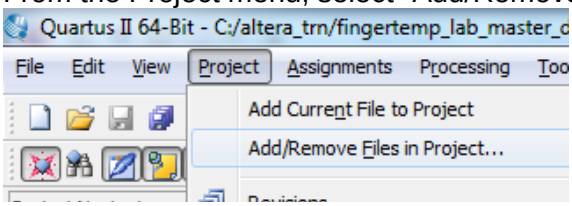



Save the Qsys system we now created as "fingertemp_adc".

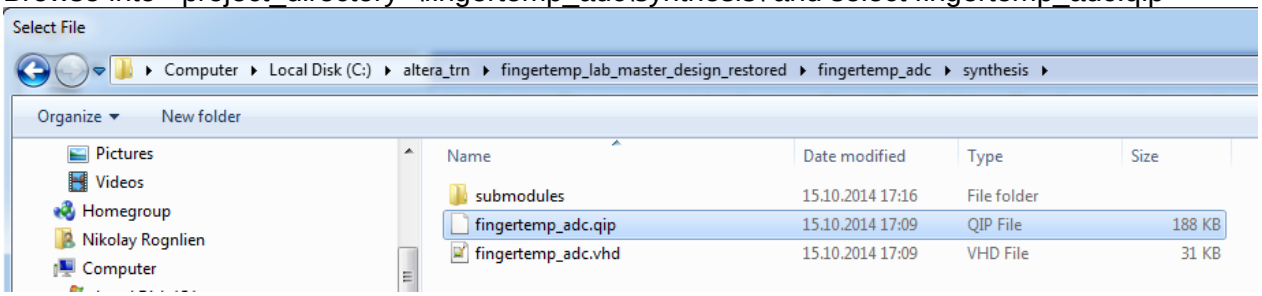
In the bottom right corner of Qsys, click "Generate HDL", select VHDL then click "Generate".
Qsys will now create synthesizable design files for the components we selected as well as the glue logic needed. (Remember we had two memory mapped masters accessing the same slave interface. Some arbitration etc. is required!)

Switch back to QuartusII GUI

You now need to add what you created in Qsys to the QuartusII project. For flexibility, Altera has created a .qip file that lists all files necessary to synthesize the fingertemp_adc subsystem. If you modify the qsys subsystem, the contents of the .qip file contents will change accordingly.
From the Project menu, select "Add/Remove Files in Project"




Browse into <project_directory>\fingertemp_adc\synthesis\ and select fingertemp_adc.qip

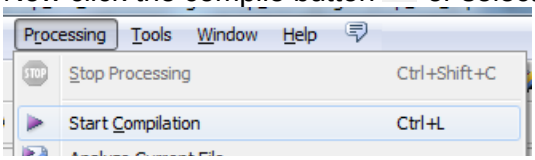


Examine the top level file called fingertemp_lab_top.
It defines an entity, which is the ins and outs of our design as well as some parameters:

```
ENTITY fingertemp_lab_top IS
   GENERIC(
      LOW_LED_BOUNDARY  : integer := 1024;
      PWM_COUNTER_WIDTH : integer := 8;
      NUM_LEDS          : integer := 8
   );
   PORT(
      SYS_CLK  : IN STD_LOGIC;                        -- inclk0
      USER_LED : OUT STD_LOGIC_VECTOR(NUM_LEDS DOWNTO 1)
   );
END fingertemp_lab_top;
```

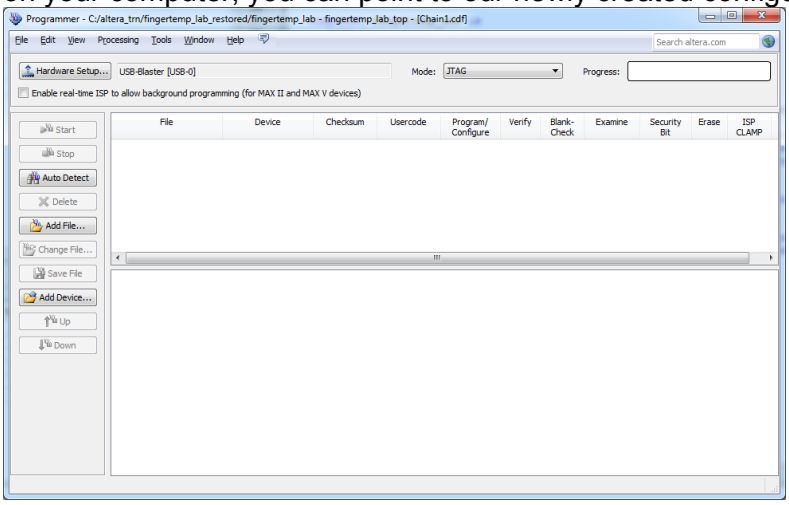
It declares some components and signals which will be used by the top level design. And after the BEGIN statement, all our cascade_pll, fingertemp_pll, fingertemp_adc and fingertemp_pwm_led components are stitched together like we saw from the RTL Viewer picture earlier in this instruction text.

Now click the compile button or select Start Compilation from the Processing menu:

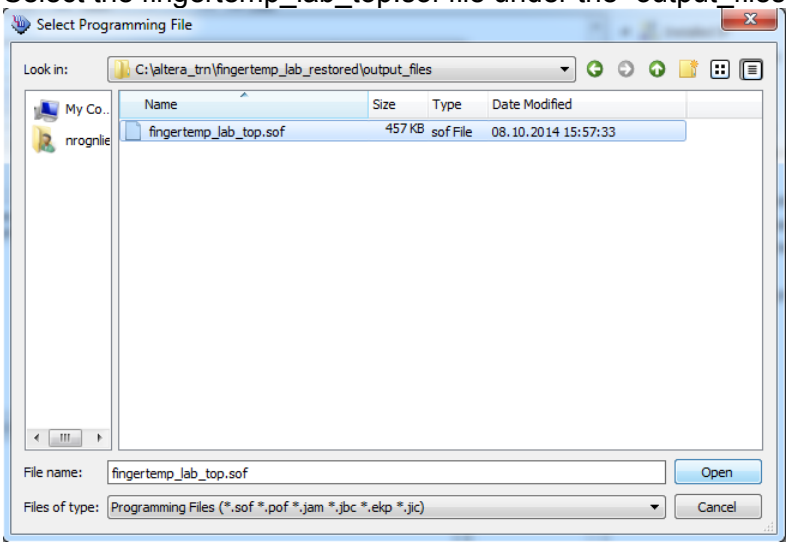



After a successful compilation(indicated by a 100% progress bar and no red errors in the Messages window, we are ready to download our design to the Max10 chip.

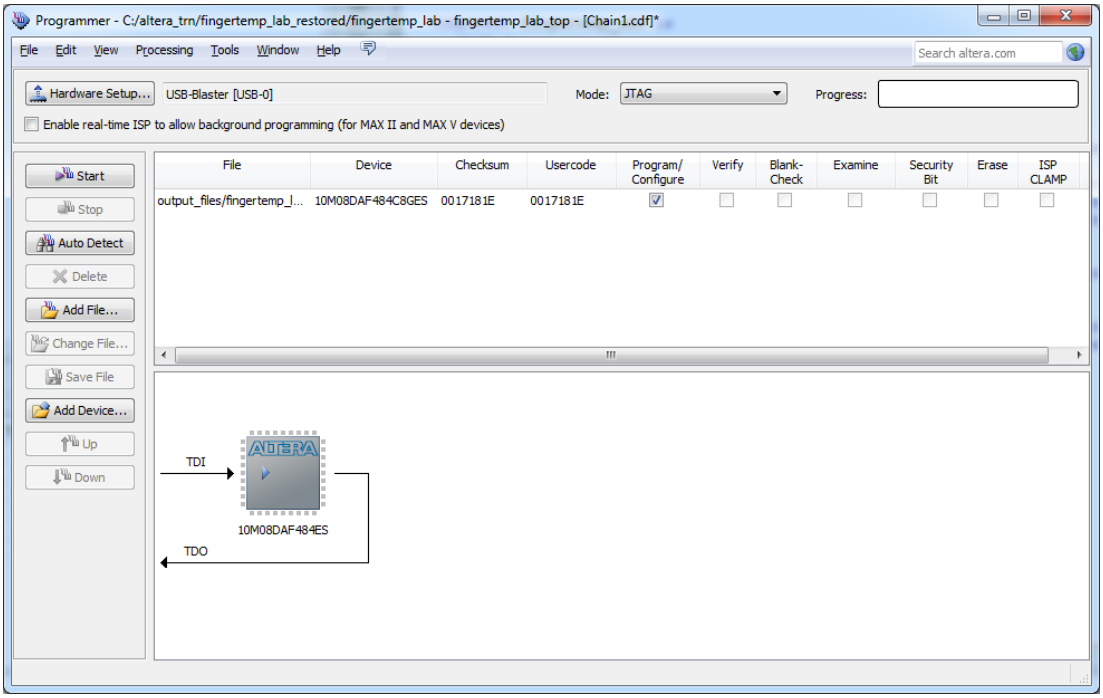
Open the QuartusII Programmer window either from the Tools menu or click from the toolbar.
If the BeMicro Max10 board is connected and the USB-BlasterII device driver has been installed successfully on your computer, you can point to our newly created configuration file by clicking the “Add File” button



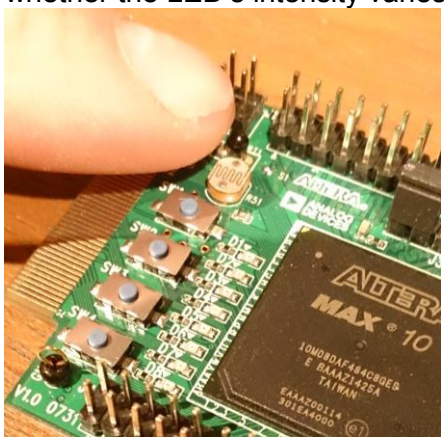
Select the fingertemp_lab_top.sof file under the “output_files”directory:




Make sure the Program/Configure checkbox is ticked and click the Start button while watching the BeMicro Max10 board…



Now put your finger on/off the thermal sensor(black, shiny, little blob in upper left corner) on the board to see whether the LED's intensity varies.




If you have time, you may explore different Generic parameters in the fingertemp_lab_top entity.

```
ENTITY fingertemp_lab_top IS
   GENERIC(
      LOW_LED_BOUNDARY  : integer := 1024;
      PWM_COUNTER_WIDTH : integer := 8;
      NUM_LEDS          : integer := 8
   );
```

The ADC puts out a value between 0 and 4095 (12 bit).
Where does the interesting finger temperature lie in this range? The LOW_LED_BOUNDARY defines at what level the LEDs start shining.

How precise does the LEDs shine around the finger temperature? The PWM_COUNTER_WIDTH is default set to 8bit which gives 256 different PWM intensities on each LED. Perhaps a bit much if you want to it narrow down to a few degrees around 37degrees C…

Have a look in the fingertemp_pwm_led.vhd file. Can you work out how it works?

## *End of Exercise*