ROCCC 2.0 Pico Port Generation - Revision 0.7.4

June 4, 2012

Contents

List of Figures

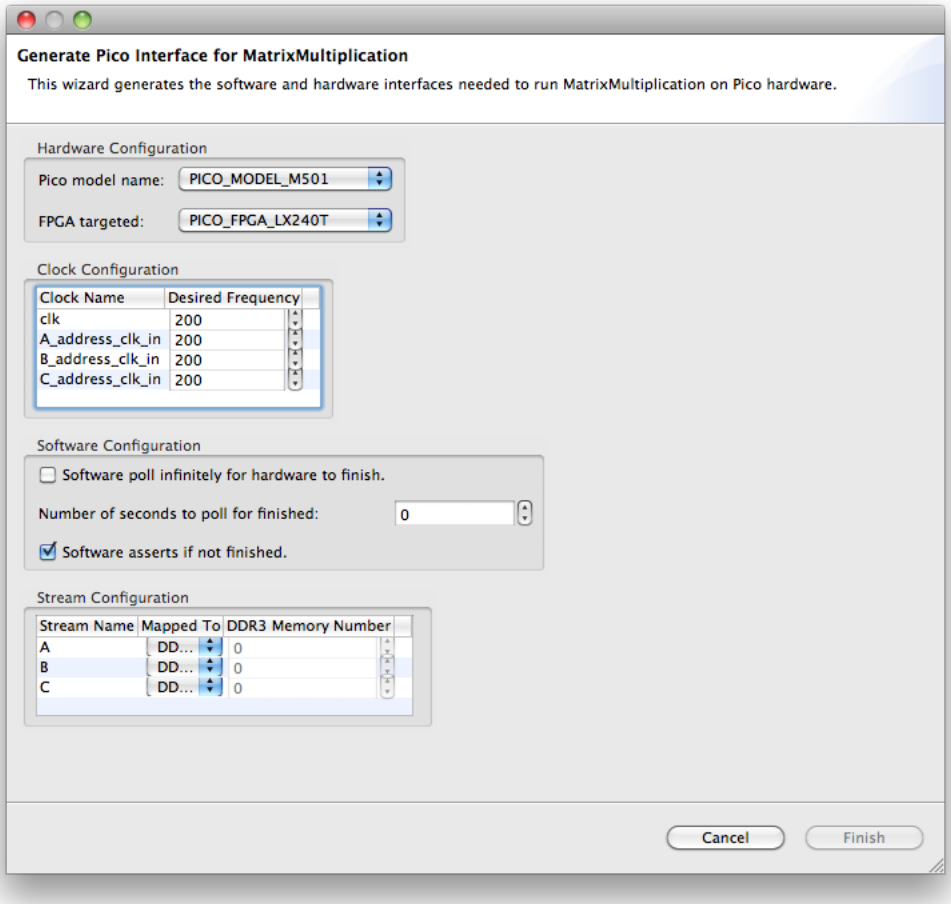

Figure 1: GUI Configuration

1 Pico Interface Generation GUI Options

When the plugin for Pico interface generation is installed, a new option to generate a pico interface will appear in the toolbar menu under "ROCCC → Generate → Pico Interface." This will generate the appropriate C++, verilog, and vhdl files necessary to run or simulate your design on the Pico Machines platform for the currently open file. The ROCCC code must be compiled previous to the generation of a Pico interface. Selecting this option will bring up a window as shown in Figure 1 that allows you to configure your interface.

1.1 Hardware Configuration

It is necessary to say exactly which target we are generating a Pico Interface for; these options allow the user to choose their target board and device. Currently, only an M501 configured with Virtex 6 LX240T's is supported. As more options become available, they will be added here and could potentially add or change the rest of the options.

1.2 Stream Configuration

Each stream identified in the C code can be configured to be placed either on the on-board DDR3 or configured as a PicoStream object. Multidimensional streams may only be configured as going through the DDR3 memory whereas infinite streams may only be configured as PicoStreams. When compiling for the Pico machines, every ROCCC stream must have only one address channel, which can be controlled via the last page of options during compilation. The total bit-width of each stream as determined by the number of data channels times the bit-width of each element cannot exceed 128.

1.3 Clock Configuration

There will be two clocks per DDR3 stream; an address clock and data clock. In addition to those, there will be one clock for the ROCCC core clock. The ROCCC core clock and DDR3 stream clocks are derived from the Pico bus PicoClk, while PicoStream clocks are derived from the PicoStream clk. Configuring a stream as a PicoStream means that the stream will run at the PicoStream clk frequency, and not the user selected frequency. Otherwise, for each clock frequency selected by the user, a Mixed-Mode Clock Manager (MMCM) primitive will be instantiated, and configured so as to match that frequency as closely as possible. Not all frequencies will be achievable; the formula is 250 MHz $*$ MULT / DIV = actual_frequency, where MULT is 5-64 and DIV is 1-128, and 250 MHz $*$ MULT is in the range 600 MHz to 1400 Mhz, inclusive. The actual frequency may be slower or faster than the specified frequency, but will minimize the difference between the two.

1.4 Software Configuration

Generating a Pico interface results in both hardware that runs on the Pico machines and software that configures and calls that hardware. When the hardware generated by ROCCC has finished executing, a hardware register is set. The generated software polls this hardware register and won't return until that register is set. You can control if this software is generated with a timeout or if the software waits indefinitely by selecting the checkbox in the Software Configuration subwindow. If you would like a guarantee that the software returns, you can select the timeout in seconds that you expect the hardware to take. Note that even if the hardware is working correctly a timeout in software will end execution. Additionally, after either the timeout has occurred or the hardware has finished, you can force a check that the hardware has finished. If this is deselected, it is possible to read data from the ROCCC hardware even if the hardware did not finish processing in order to see the current status.

2 Using the Generated Firmware

2.1 Copy Example

In order to synthesize and run the hardware generated from ROCCC, you will need to copy the latest example project from your Pico distribution into a local directory and overwrite some of the files with the ones generated by ROCCC. The sample projects should be located on your Pico machine under /usr/src/picocomputing-X/samples, where X is the latest version number installed on your machine. Once the directories have been set up you will need Xilinx ISE in order to synthesize and generate the appropriate bit file.

If you mapped any streams onto DDR3, you will need to copy an example that utilizes the DDR such as DDR3_MovingAverage. If you mapped your streams only onto PicoStreams, you will need to copy an example that does not initialize the DDR, such as StreamLoopback128. Using an example that uses DDR3 with a ROCCC core that does not use DDR3 will mean that the constraint file ./firmware/ISE_m501lx240/source/M501_LX240T_DDR3.ucf will cause errors; avoid this by either copying over the correct example, or removing the constraint file when not using DDR3.

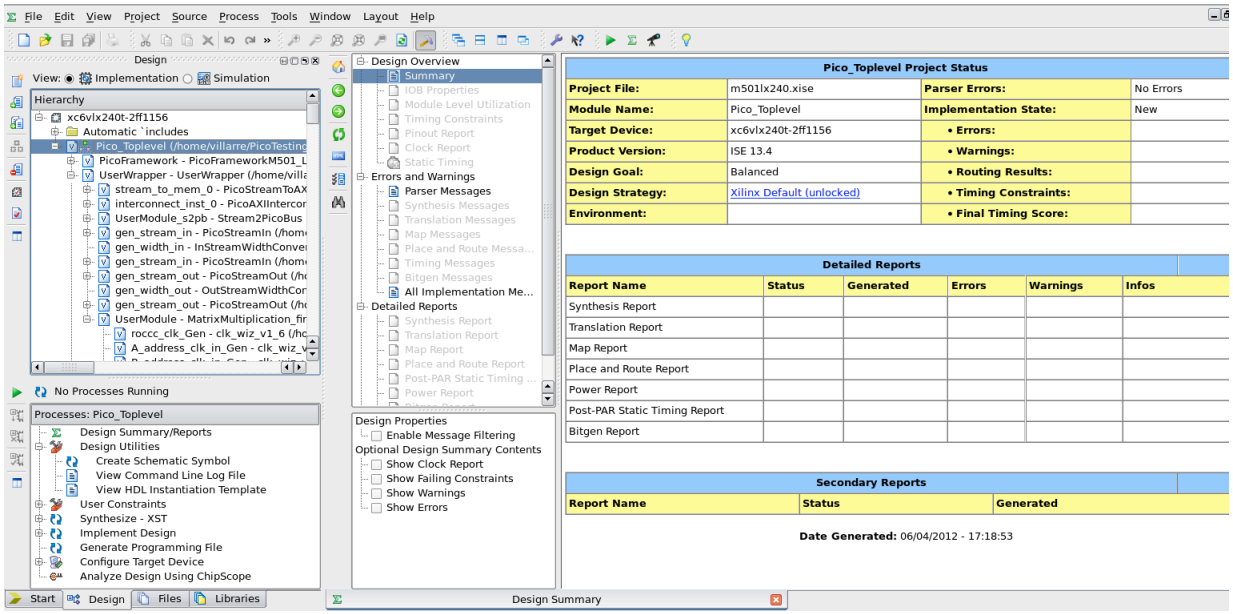

Figure 2: ISE after loading the project with the additional files for the MatrixMultiplication example

2.2 Add Required Files

The project directory should have two subdirectories, firmware and software. Under the firmware directory should be a subdirectory specific to the platform you are targeting (currently the M501 is supported), in this subdirectory you will find the m501lx240.xise file which you need to open with ISE. In ISE, add all of the *.v and *.vhdl files ROCCC generated in the PicoInterface folder. Additionally, replace the PicoDefines.v file located under firmware/ISE_m501lx240/ with the PicoDefines.v file generated by ROCCC.

2.3 Generating a Bitfile

Once the necessary files have been added, the project should be preconfigured to generate a bitfile for the appropriate platform, which is done through ISE as shown in Figure 2. Select Pico_Toplevel as your top module and generate a bitfile by double clicking "Generate Programming File" in the Design window.

If timing errors are encountered, try reducing the clock frequency of the stream clocks and ROCCC core clock; regenerate the Pico Interface from the ROCCC GUI with lower numbers specified in the clock frequency section and reimport the generated files.

The generated bitfile can either be loaded into the FPGA through the Pico Purty tool or through the generated software.

3 Using the Generated Software

ROCCC generates software functions that initialize the hardware and start the hardware with the appropriate data, but these functions need to be called. While we have included in this distribution a complete working system for each of the provided examples, in general it is the user's responsibility to call the generated functions from their own code.

```
template<class T>
T* convertToLowerDimensionArray(T** array, int rows, int cols, T* ret = NULL)
{
  if( ret == NULL )
  {
    ret = new T[rows*cols];
  }
  for(int i = 0; i < rows; ++i)
    for(int j = 0; j < cols; ++j)
      ret[i*cols+j] = array[i][j];
  return ret;
}
```

Figure 3: C++ code to down-convert a two dimensional array into a one-dimensional array to be passed to the hardware

3.1 Initializing Hardware

There are two options when initializing the hardware; either an initialization function that is generated by the ROCCC GUI can be called, or the user can initialize the hardware themselves. Regardless, the hardware MUST be initialized before calling any of the ROCCC generated software calls. The initialization function created by ROCCC will have the signature "PicoDrv* initializeHardwareForX(char* bitfileName)" where X is the name of the ROCCC core; this call finds an empty M501 card, loads the card with the specified bitfile, and returns a pointer to the PicoDrv object that represents that connection to the card. The parameter bitfileName can be passed in as NULL; doing so searches for a card already configured with the firmware that matches the software's core ID (generated by the ROCCC GUI) and returns the first matching card. If the user wishes, they may load the bitfile themselves; if they do so, they should make sure to call the _X() version of the ROCCC generated software calls, and not the X() version (which can only be utilized with the PicoDrv object created with the initializeHardwareForX() call).

3.2 Stream Sources

Both DDR3 streams and PicoStream streams must be passed to the hardware call as single dimensional arrays; this means that any multidimensional arrays have to be converted to single dimensional arrays before passing them to the hardware. This must be done in row order; if the original array was A[10][10], then you can make a single dimensional array A_single[100] by setting A_single[x*10+y] = A[x][y]. The code shown in Figure 3 can be used to easily convert arrays down to a single dimensional array.

Similarly, the results coming back from hardware are single dimensional arrays and must be converted back to the appropriate dimensionality. The code in Figure 4 snippet can be used to convert single dimensional arrays back up to multidimensional arrays.

When creating streams, it is important to keep in mind the size of the memory array ROCCC is expecting; if your access window is a 3x3, then you have to create an array that is 2 larger in each dimension, to allow for this access window. For example, if the code in Figure 5 is compiled with ROCCC, when calling the hardware you must pass in an array with dimensions 12x12.

Additionally, DDR3 streams are set by the hardware before starting the ROCCC core, while PicoStreams are streamed to the ROCCC core while it is running; for both non-infinite DDR3 streams and PicoStreams, this is handled by the main software core call. For infinite streams, this must be handled by the user, by calling additional stream access functions. These additional stream functions are generated by the ROCCC GUI, and are called writeInputDataX and readOutputDataX (where X is the name of the stream). These functions have a limit of how much data can be passed in any given call; because writing to a stream is a blocking write, trying to write too much data will stall the system from back pressure. Instead, it is necessary

```
template<class T>
T** convertToHigherDimensionArray(T* array, int rows, int cols, T** ret = NULL)
{
  if( ret == NULL )
  {
    ret = new T*[rows];
    for(int i = 0; i < rows; ++i)
      ret[i] = new T[cols];
  }
  for(int i = 0; i < rows; ++i)
    for(int j = 0; j < cols; ++j)
      ret[i][j] = array[i*cols+j];
  return ret;
}
```

Figure 4: C++ code to up-convert single dimensional output into two dimensional arrays

```
for (i = 0 ; i < 10 ; ++i)
{
  for (j = 0 ; j < 10 ; ++j)
  {
     // This will access elements 0,0 through 11,11 requiring a 12x12 input
     C[i][j] = A[i][j] + A[i+2][j+2] ;
  }
}
```

Figure 5: ROCCC code that accesses a window

to write and read chunks of data until the final amount of data is processed; this must be handled by the user. Each reading and writing function will only read or write as much data as there is space available, without blocking, and then returns the number of elements read or written. Because the ROCCC generated software call will stall while waiting for the hardware to finish (see synchronization options above), and only then read the output scalars, if output scalars are used, infinite streams should not be used with output scalars.

3.3 Invoking the Generated Software

There are two possible options when calling the generated software; there is a standard call, X(), and a more versatile function call _X(), which takes an additional PicoDrv* argument to select the card we are using. When the standard call X() is used, the PicoDrv object used is the same one returned by the initializeHardwareForX(); this pointer object is a global shared by the various ROCCC generated software calls. An example of the two forms follows:

```
void VectorReduction(ROCCC_int32* A, ROCCC_int32 N,
                     ROCCC_int32 cumulative_init, int& sum_out) ;
void _VectorReduction(PicoDrv* pico, ROCCC_int32* A, ROCCC_int32 N,
                      ROCCC_int32 cumulative_init, int& sum_out);
```

These two functions do the following:

- turn on ROCCC reset, and turn off ROCCC inputReady
- send input scalars to hardware
- write DDR3 streams to DDR3 (if applicable)
- turn off ROCCC reset, turn on ROCCC inputReady
- poll ROCCC done for specified number of seconds, or infinitely, breaking when ROCCC done goes high
- while polling, read and write PicoStreams (if applicable) in chunks
- read output scalars from hardware
- read DDR3 streams from DDR3 (if applicable)
- return to calling code

After returning to the calling code, any DDR3 streams will be filled with 1-dimensional data calculated from the ROCCC core; to convert this back to a multi-dimensional array, use the function snippet "convertToHigherDimensionArray()" listed above.

