picoTools Product Brief

picoTools

1. Introduction

picoTools are the picoChip tool chain for mapping designs on to picoArrays. The picoArray is a massively parallel processor array targeted at DSP solutions. This approach allows DSP systems to be specified and verified under one unified design environment, (picoTools). It enables much improved system integration times over conventional/legacy DSPs due to the picoArray architecture's inherent determinism. Ultimately it allows a reduced bill of materials due to the picoArray's cost efficient delivery of MIPS. The picoTools flow allows functions formally implemented across ASIC/FPGA and DSP to be implemented within one processing fabric.

2. picoArray programming

The basic programming model is to describe the system's processes or tasks in ANSI C or assembler, with a network of fixed connectivity and of specified bandwidth. This is a very natural way to describe DSP systems. This is shown in figure 1.

The interconnect of the processes is defined using the structural subset of VHDL, a language designed to specify connectivity.

picoTools assign processes to fixed resources at compile time. Similarly the interconnect is pre-assigned to slots of the fabric. In this way there is no run time scheduling of resources. This means that the performance of a sub-system is deterministic whether standalone or integrated into a full system. This hugely reduces system integration and verification times.

3. picoTools

The function of picoTools is to implement, debug and verify systems on picoArrays. It offers an integrated compiler and assembler, tools to map the design to picoArrays, and a fully featured parallel debugger. This debugger can equally well be used with a program executing on hardware or on the cycle accurate simulator.

The tool chain development stages are to pass the entire design through analysis, which extracts its connectivity. Then the compiler and assembler act on the actual processes themselves. The system can then be interactively partitioned across multiple picoArrays. For each picoArray the design's processes and signals are automatically assigned onto the array and fabric. The result of this assignment or placement can then be used to either run the design directly on hardware or to create a simulation.

The debugger has been architected to work on parallel systems. It allows unlimited processes to be debugged simultaneously. The whole system can be single stepped together. Breakpoints or asserts can be added to any process and cause the whole system to halt. Each process may be viewed as source code, disassembly or one interleaved with the other. The system can be viewed as a hierarchical or flat graphical network, where the state of each process is color coded on the GUI. All of these features provide an excellent debugging environment.

The tool chain supports the debugging of hardware through a JTAG connection using picoICE.

All of the tools can be controlled either using GUIs or can be scripted in Tcl/Tk. This makes the interfaces fully extensible, so for example user defined GUIs can be created for high level application specific debugging.

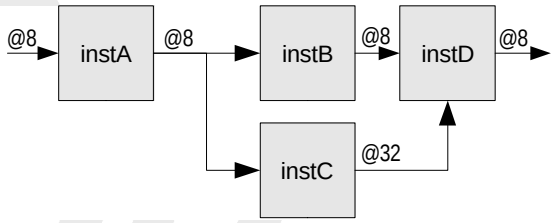


Figure 1 Programming model

4. Supported Platforms

picoTools are available for Linux RedHat 8.0 and Enterprise 3.0 platforms.

Processes are specified in ANSI C or assembler.

The process IO and connectivity are specified in structural VHDL

```
entity NCO is                          Specify process IO
  generic (FREQ_BASE : integer16);
  port (oscillator:out blocking integer16pair@16);
end entity NCO;

architecture C of NCO is               Specify process
begin ANY                              function in C
CODE
int main() {
  integer16pair out_phase;
  int sine_phase, sine;

  while (1) {
    if (quadrant & 1)
     sine_phase = ((1 << 6) - 1) - sine_phase ;
    sine = sine_lookup[sine_phase];
    // invert sine?
    if (quadrant & 2) sine = -sine;
    out_phase.el1 = 0;
    out_phase.el2 = sine;
    putoscillator(out_phase);
  }
}
ENDCODE
end NCO
```

```
entity NCO_2 is                        Specify process IO
  generic (FREQ_BASE : integer16);
  port (oscillator:out blocking integer16pair@16);
end entity NCO;

architecture C of NCO_2 is
begin ANY
CODE
  loop
    ADD.0 phase, FREQ_BASE, phase
    LSR.0 phase, 8, sine_phase
    AND.0 sine_phase, 16#3f#, sine_phase
    AND.0 [LSR phase, 14], 1, scratch
    if (NONZERO) then
      SUB.0 63, sine_phase, sine_phase
    end if

    LSL.0 sine_phase, 1, AP
    LDW (AP), sine
    PUT R[5:4], oscillator
  end loop                             Specify process
ENDCODE;                               function in ASM
end NCO_2
```

Compile

The design is compiled and assembled as a whole.

Compile

ANSI C is compiled using the gcc compiler

picoPartition

For designs using multiple picoArrays it can be interactively partitioned.

The project build can all be controlled from scripted or GUI of picoDeveloper

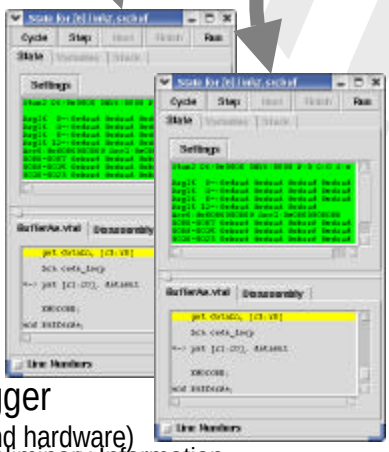
picoPlastic

The processes are automatically assigned to processors on the array, and slots on the bus resource allocated.

All tools can either be controlled through GUIs or can be script controlled using Tcl/Tk

Designs can then be targeted at the cycle accurate simulator or onto hardware, and debugged with an identical interface.

The debugger is fully parallel. The whole system can be single stepped, or stopped at breakpoints.



The design can be viewed as a graphical network using the Design Browser at all stages of the flow

picoDebugger

(Simulation and hardware)

Programming model and Tools flow details

Programming

Specify process IO

```
entity NCO is
  generic (FREQ_BASE : integer16);
  port (oscillator:out blocking integer16pair@16);
end entity NCO;
```

Specify process function in C

```
architecture C of NCO is
begin ANY
CODE
int main() {
  integer16pair out_phase;
  int sine_phase, sine;

  while (1) {
    if (quadrant & 1)
     sine_phase = ((1 << 6) - 1) - sine_phase ;
    sine = sine_lookup[sine_phase];
    // invert sine?
    if (quadrant & 2) sine = -sine;
    out_phase.el1 = 0;
    out_phase.el2 = sine;
    putoscillator(out_phase);
  }
}
ENDCODE
end NCO
```

Figure 1 Code example

To build and implement a DSP system on a picoArray the system is hierarchically decomposed into a set of communicating tasks or processes, with a fixed connectivity of a specified bandwidth. Figure 1 shows an example of code for such a process. The actual function is defined inside the `CODE ENDCODE` section. This can be in either ANSI C or assembler. The code that encapsulates this is in the structural subset of VHDL93. It is used to specify the interfaces of the block and at higher levels of the hierarchy to describe how processes interconnect.

This dataflow view of a DSP system is a very natural way of specifying it, and maps well to the picoArrays stream based processing.

The hierarchical decomposition results in individual tasks that are assigned to processors by the tool chain; one task is mapped to one processor. In this way each process gets a fixed resource on which to run. There is no need for an operating system on a picoArray, since there is no dynamic scheduling of processes to be done.

Providing fixed resources for both tasks and their interconnect means that designs have deterministic behaviour, so as a system is integrated its parts will continue to operate exactly as they did when verified alone. This will not be true of a dynamically scheduled system. This is a huge advantage in verification intense applications.

Compilation

A system is read into the elaboration and compilation step as a whole. The configuration of a system build can be controlled by the use of VHDL style libraries and "Use" constructs in the code. This allows encapsulation of function and control of the build's function.

The elaboration extracts the process connectivity from the system, and then farms out the compilation and assembly of the processes themselves to the compiler and the assembler.

The compiler is a port of the industry standard gcc compiler. The C initiates communication on signals with the calling of to a single argument built-in.

picoPartition

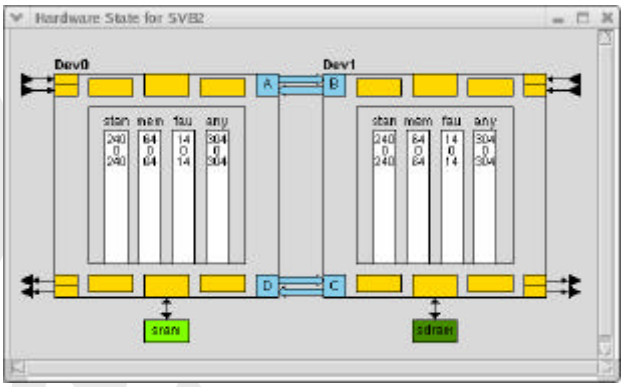


Figure 2 picoPartition GUI

picoArrays are designed to be cascadable. The cascading interfaces are an extension of the processor fabric allowing multiple devices to operate seamlessly as a single system.

The picoPartition tool allows a system to be spread over multiple picoArrays.

The configuration of a board in terms of number of chips and their arrangement is specified to the partitioning tool. The system itself is then read in from the compilation phase and can be interactively split amongst the devices. The tools automatically deal with the setup and control of the device interfaces.

Figure 2 shows the picoPartition GUI for a two-device system. The amount of resource remaining on the devices during the partition is shown.

picoPlastic

Figure 3 picoPlastic GUI

picoPlastic is responsible for the assigning of tasks to processors and allocating the task's connectivity to time division slots of the picoArrays fabric. The stage is known as "place and switch". In effect it is performing the static scheduling of the fabric.

Each device is "Placed and Switched" individually in what is an automatic, and largely "push button", step for the user.

The result of this step is a configuration file or near boot image for a picoArray.

picoDebugger

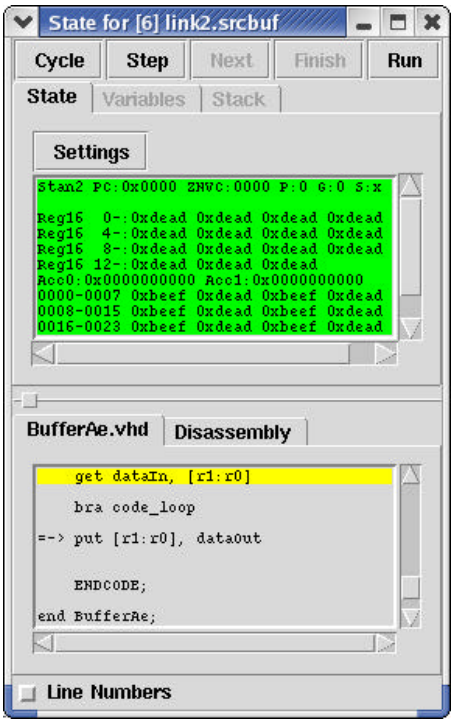


Figure 4 picoDebugger instance GUI

picoDebugger is a front end to systems running on either the cycle accurate simulator and those running on actual picoArray hardware. picoICE can be used to debug systems via a JTAG interfaces.

The debugger provides all of the normal features that would be expected. The source of either C or assembler can be viewed and stepped though for each process, and its variables, memory and registers viewed.

Additionally as a parallel debugger many processes can be debugged simultaneously. As one process is stepped all of the others progress, when one process hits a breakpoint all of the others also halt - even over multiple devices.

To reinforce the system centric view of the tool suite designs can be viewed in the Design Browser. This can hierarchically display the tasks and their interconnect. When used in the debugger it can additionally display the state of each process to show if it is running or stalled.

Figure 5 shows the Design Browser GUI

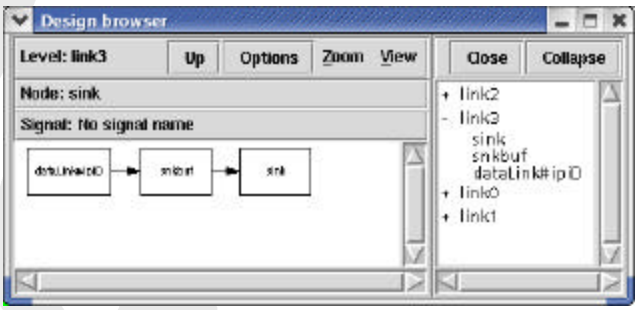


Figure 5 Design Browser GUI

Products

PC5100	License for picoTools, per seat
PC5200	License for picoTools, per site
PC5100-temp	License for picoTools, 30 day evaluation
PC5300	picoICE JTAG debugger

For more information contact:

info@picochip.com or visit www.picochip.com