How to run a C or Assembly program

1. Put C code in ‘piton/verif/diag/c’
    1. It can go in any folder or a new folder, just remember which
2. At top of C code put
    1. #include "libc.h"
    2. You do not need to include boot.s
    3. In order to use print statements weird things are needed
3. Create an assembly file in piton/verif/diag/assembly/c/
    1. #define USE_STACK
    2. #include "c/template_mt.s"
    3. MIDAS_CC FILE=<*where you put your c code in step 1*> ARGS=-O2 -S
4. cd $MODEL_DIR
5. Create a <list name>.txt file and put a full path to the assembly test inside
6. perl newCompScript.pm <list name>.txt
    1. protosyn -b vc707 -d system --uart-dmw ddr
7. Open Vivado
    1. Select the project that was just generated in build/vc707/system/vc707_system/vc707_system.xpr
    2. Put it in the board
8. pitonstream -b vc707 -f <list name>.txt

For Assembly simply write an assembly program at step 3, skipping 1 and 2

Important Files

- $PITON_ROOT/piton/tools/perlmod/Midas/3.30/bin/midas
    - The assembler for OpenPiton, invoked by pitonstream to turn make assembly files into data that is loaded onto the board
- $PITON_ROOT/piton/tools/src/proto/fpga_lib.py
    - This file contains the command which launches Midas, to which you can add flags
    - “-x_tiles=2 -y_tiles=1” makes the assembler recognize the number of core tiles (must match hardware)
    - “-midas_args='-DCIOP -DTHREAD_COUNT=2’” Sets the total number of threads you want
    - “-midas_args=’-DTHREAD_STRIDE=2'” Sets the number of cores you want to run on
    - Ex: cmd = "sims -sys=manycore -novcs_build -novera_build -midas_only -x_tiles=2 -y_tiles=1 -midas_args='-DUART_DIV_LATCH=0x%x -DFPGA_HW -DCIOP -DNO_SLAN_INIT_SPC -DTHREAD_COUNT=2 -DTHREAD_STRIDE=2' %s" % (uart_div_latch, tname)
- $PITON_ROOT/piton/design/xilinx/pyhp_setup.tcl
    - Specify some pieces of the hardware specification
    - Use this to adjust number of cores (x * y)
    - Do not mess with the caches, this causes it to fail

How to Change the Number of Cores in Hardware

- $PITON_ROOT/piton/design/xilinx/pyhp_setup.tcl
    - These lines control the number of cores
    - set ::env(PTON_X_TILES) 2
    - set ::env(PTON_Y_TILES) 1
    - set ::env(PTON_NUM_TILES) 2
        - This is x*y

How to Change the Number of Cores in Software

- $PITON_ROOT/piton/tools/src/proto/fpga_lib.py
    - Go to the following function
    - def runMidas(tname, uart_div_latch, flog):
    - In the “cmd=” line change the following:
        - These should match the hardware
            - -x_tiles=
            - -y_tiles=
        - This controls how many threads will run in parallel
            - -DTHREAD_COUNT=
        - This controls how many cores you want the threads to run on. (This should NEVER be greater than DTHREAD_COUNT)
            - -DTHREAD_STRIDE=

Assembly Test Generation

```
#define ADDR1 0xfff0c2c000
#define ADDR0 0x9a00000000
#include "boot.s"
.text
.global main
main:
        setx active_thread, %l1, %o5
        jmpl    %o5, %o7
        nop

!
!       Note that to simplify ASI cache accesses this segment should be mapped VA==PA==RA
SECTION .ACTIVE_THREAD_SEC TEXT_VA=0x0000000040008000
   attr_text {
        Name = .ACTIVE_THREAD_SEC,
        VA= 0x0000000040008000,
        PA= ra2pa(0x0000000040008000,0),
        RA= 0x0000000040008000,
        part_0_i_ctx_nonzero_ps0_tsb,
        part_0_d_ctx_nonzero_ps0_tsb,
        TTE_G=1, TTE_Context=PCONTEXT, TTE_V=1, TTE_Size=0, TTE_NFO=0,
        TTE_IE=0, TTE_Soft2=0, TTE_Diag=0, TTE_Soft=0,
        TTE_L=0, TTE_CP=1, TTE_CV=1, TTE_E=0, TTE_P=0, TTE_W=1
        }
   attr_text {
        Name = .ACTIVE_THREAD_SEC,
        hypervisor
        }
!
!
!

.text
.global active_thread

!   We enter with L2 up and in LRU mode, Priv. state is user (none)
!
!
active_thread:
        ta      T_CHANGE_HPRIV          ! enter Hyper mode
        nop
th_main_0:
```

Put your assembly code here

If you want to be able to print uart, you will need your test to include this code at the top.

Our Print Function

```
        .align 4
        .global uart_reg_print
uart_reg_print:
        mov %o0, %l2
        mov 0x4, %l1
loop_print:
        and %l2, 0x7, %l4
        add %l4, 48, %l4
        mov %l4, %o0
        call uart_char_print
        nop
        srl %l2, 3, %l2
        cmp %l2, 0
        bne loop_print
        nop
        retl
        nop


        .align 4
        .global uart_char_print
uart_char_print:
        setx ADDR1, %l6, %l5
        mov %o0, %l3
        stb %l3, [%l5]
        setx 0x5420,%l6,%l4
loop_10:
        dec %l4
        cmp %l4, 0
        bne loop_10
        nop
        retl
        nop
```

Uart_reg_print will print out the contents of the %o0 Register in reverse oct format.

Uart_char_print will print out the ascii character in %o0

Running Multiple Threads

```
.text
.global active_thread
.global th_main_0
.global th_main_2
!
!   We enter with L2 up and in LRU mode, Priv. state is user (none)
!
!
active_thread:
        ta      T_CHANGE_HPRIV          ! enter Hyper mode
        nop
        setx LOCK_ADDR, %l6, %i4
        clr [%i4]                       !sets initial mutex to 0
        setx SHARED, %l6, %g5
        clr [%g5]
        setx FINISH, %l6, %g6
        clr [%g6]
        ta T_RD_THID
        nop
        mov %o1, %l7
        cmp %l7, 0
        be th_main_0
        nop
        cmp %l7, 2
        be th_main_2
        nop
        ta T_BAD_TRAP
        nop

th_main_0:
        setx 150, %l6, %i3
```

For each you want to create you will need to:

- Declare a global
- Branch to it after running T_RD_THID
- Add to the function after

The th_fork function doesn't seem to work. So, this is the manual way of doing it.

Current Work on Mutexes

Both of these do not work right now. The first one was created by us. The second implementation was created by them. The membar function seems to be the issue in theirs. So, ours works but has bugs.

Important Notes

- Most of the time, you will have to resynth after any major changes to your tests. This will take about 45 minutes
- Print does not currently work in C tests
- Sometimes even if you only added lines of assembly instructions, you could be timing out or failing a test. Resynthing might fix the problem.
    - If you don't you might run into errors that are caused by the code overwriting parts of itself.
- Some of their tests sort of workish
- In order to avoiding having to resynth after changing a small part of a test, follow the following rules.
    - If you add a read or write to memory in a location not in the test you will probably need a resynth
    - If you add a new test to you list file, you will need a resynth
    - If you add a new function you will occasionally need a resynth

Register Information:

r0-r7	g0-g7	Global Registers They are shared between threads, not cores. g0 is always 0.
r8-r15	i0-i7	Input Registers Local to a function call.
r16-r23	l0-l7	Local Registers Local to a function call.
r24-r31	o0-o7	Output Registers These are read in as the input of functions.

Useful Links:

- http://www.cs.northwestern.edu/~agupta/_projects/sparc_simulator/SPARC%20INSTRUCTION%20SET-1.htm
    - Sparc ISA
- http://venividiwiki.ee.virginia.edu/mediawiki/images/b/b4/OpenSPARCT1_DVGuide.pdf
    - An OpenSparc guide
- https://encrypted.google.com/patents/CN102708090A?cl=en
    - Some documentation on how threads work
- http://www.oracle.com/technetwork/systems/opensparc/t1-01-opensparct1-micro-arch-1538959.html
    - Microarchitecture spec

OpenSparc Architecture

Each SPARC core has the following units:
1. Instruction fetch unit (IFU) includes the following pipeline stages – fetch, thread selection, and decode. The IFU also includes an instruction cache complex.
2. Execution unit (EXU) includes the execute stage of the pipeline.
3. Load/store unit (LSU) includes memory and writeback stages, and a data cache complex.
4. Trap logic unit (TLU) includes trap logic and trap program counters.
5. Stream processing unit (SPU) is used for modular arithmetic functions for crypto.
6. Memory management unit (MMU).
7. Floating-point frontend unit (FFU) interfaces to the FPU.

IFU
Two instructions are fetched each cycle, though only one instruction is issued per clock, which reduces the instruction cache activity and allows for an opportunistic line fill. There is only one outstanding miss per thread, and only four per core. Duplicate misses do not issue requests to the L2-cache.

Decoder Location
Design/chip/tile/sparc/ifu/rtl/sparc_ifu_dec.v

Future Work

(or, where to start working on this)

- Update to the latest version of OpenPiton
    - The “official” mutex implementation is not supported on the version we are using, hence why “membar” does not work
    - You will have to edit the number of cores in software and hardware again
    - You will have to install a gcc cross compiler, which they provide, and set the root variable in the setup file to point to this (there may be other changes needed to the environment variables in this script)
    - Hopefully it will just work out of the box other than these changes
- Make char print slightly more robust
    - Currently it has to spin for a huge (arbitrary) number of cycles to work correctly
    - It seems like there should be a way around this based on the properties of the UART serial line
- Get char print working in C
    - This currently does not work because the compiler does not include the parts of the code that are necessary for print (address setup and transition to hypervisor mode)
    - This might require looking into how the assembler and the linker work, unfortunately, as simply adding those sections in preprocessing did not work
- Get mutexes working in C
    - This will probably just work by adding a C wrapper for the assembly version
    - Assuming their mutexes work
- Get threads working in C
    - Since there is no “thread fork” functionality, a C wrapper for how threads work in this ISA is slightly more complicated
- Add scratch pad memory to hardware
    - Simply pick part of the memory map and make a splitter that maps that section of memory to the scratch pad
    - The scratch pad memory should bypass the caches and be implemented in BRAMs
    - Possibly look into how the UART connection is done, as this is also just a MMIO address
- Make the system work with C++
    - Good luck