Hardware Thread Interface Design Document

Purpose and Overview
... blah blah blah ... yadde yadde yadde ...

Example Files
To illustrate how the HWTI works, example files are placed within my home directory /
users/eanderso/Documents/hybridthreads/hwti.
• opb_hwti.vhd: Bus connection for the HWT.
• user_logic_hwti: Stub file for HWTI. Only performs handshake protocol with HWTUL.
• user_logic_hwtul.vhd: Example file for the HWTUL. Entity exits as soon as RUN is issued.
• hwt_testbench.vhd: Test bench to test interaction between HWTUL and HWTI.

System Level Application Programming Interface
The system level API consists of a set of five memory mapped registers. The register names are
thread_id, verify, command, status, argument, and result. The description of each of these
registers, plus the protocol on how to use them, are below.

All registers, since they are accessed directly off of the PLB, are 32 bits.

thread_id

Overview
The thread_id register tells this thread what its thread id is. The thread id is assigned by the
system at runtime. Specifically, when a thread is created, the system asks the Thread Manager
for a thread ID, the system then assigns the thread ID to this register.

The thread_id register is both readable and writable.

Protocol
On system start up, and after a reset, the thread_id is set to 0. When the thread_id register is
written to, the status changes from NOT_USED to USED. The thread_id may be written to only
when the status register reads NOT_USED. With all other statuses, writing to this register has no
effect. Bits 24 to 31, of the system bus data lines, are used to set the thread_id of the HWT.

The thread_id register may be read from at anytime. The read operation does not have any side
effects.

The thread_id must be set prior to RUN being issued to the command register.
**verify**

**Overview**

The verify register is read by the system to indicate that a hardware thread exists at this address location.

**Protocol**

... coming to a design document near you soon ...

**command**

**Overview**

The command register is written to by the system to tell the HWT to RUN, RESET, STEP, or IDLE. A RUN command serves two purposes. First to tell the HWT to start executing, second, if the HWT was waiting on a mutex, to wake up and check the mutex. The RESET command tells the HWT to reset all variables (both in the HWTI and HWTUL) and return to a NOT_USED status (see status register).

The STEP and IDLE command are for debugging. The IDLE command tells the HWT to stop what it is doing and remain in its current state without changing any of its internal state. The STEP command tells the HWT (specifically the HWTUL) to proceed to the next state and then IDLE.

**Protocol**

A RUN may be issued to the HWTI only if the status register is either USED, BLOCK, or IDLE. Issuing a RUN at any other time has no effect on the HWT.

Issuing a RUN while the status is USED changes the status to RUNNING. This will also change the user_status from RESET to RUN, triggering the HWTUL to start executing its state machine.

Issuing a RUN while the status is BLOCK, tells the HWTI to recheck the operation causing the block (mutex lock, or interrupt associate), if successful change the user_status from ACK to RUN, triggering the HWTUL to continue on with its state machine.

Issuing a RESET at anytime sets the status register to NOT_USED, the thread_id register to zero, and the user_status register to RESET. The HWTUL is responsible for resetting any variables it may use. To insure the HWT is in an initialized state, the system should RESET at start up. The system must also issue a RESET if, after the HWT exits, the system wants to reuse the HWT component as a new thread.

The IDLE command is used to temporarily halt the HWTUL's state machine. This should only be used by the system for debug purposes. To restart the HWTUL's state machine, the system must issue a RUN or STEP command. IDLE will only have an effect on the HWT if the status is either RUNNING or BLOCKED. When issued, the IDLE command changes the status to IDLING.

The STEP command is used to advance the HWT to the next state (assuming the conditions exist to advance to the next state) and then return to IDLE. STEP may only be issued when the status register is IDLING. After STEP, the status remains at IDLING.
When an IDLE command is issued, the HWTI sets the user_status to PAUSE, and the status register to IDLING. The HWTUL's state machine must remain in its current state until the HWTI changes the user_status back to RUN or ACK (the user_status prior to idling the HWT). When STEP is issued, the HWTI changes the user_status to either RUN or ACK for one clock cycle, and then returns the user_status to PAUSE.

The command register may be read from at any time, returning the last command the HWT received. This call has no side effect.

The binary values of each command are as follows:

- RUN (0001)
- RESET (0010)
- IDLE (0100)
- STEP (1000)

Bits 28 to 31, of the system bus data lines are read to determine the value of the command.

NOTE: there is no current way for the HWTUL's state machine to communicate to the HWTI its state. Does there need to be an additional register set?

### status

#### Overview

The status register is a read only register, indicating to the system the state the HWT is in. It should only be used for debugging purposes. The possible states the HWT may be in are RUNNING, BLOCKED, IDLING, EXITED, USED, NOT_USED.

#### Protocol

The HWTI will report each state for the following conditions. Binary values are in parenthesis.

- **NOT_USED** (0000 0000): This is the state of the HWT on system start up and after a RESET command. No other commands have been issued.
- **USED** (0000 0001): This is the state after the thread_id register has been populated, but before a RUN command has been issued.
- **RUNNING** (0000 0010): The thread_id register has been populated, the system issued a RUN command, the HWT is not waiting on a mutex or other blocking type of operations, and the HWT has not exited. Generally means that the HWTUL is executing its state machine (doing useful work).
- **BLOCKED** (0000 0100): May transition to a BLOCKED state from a RUNNING state. Occurs when the HWTUL issues a REQUEST_LOCK operation, and the HWTI is waiting to obtain the lock. Once the lock is obtained, status transitions back to RUNNING. Generally means the HWT is waiting to obtain a mutex.
• EXITED (0000 1000): The HWT will transition to this state after the HWTUL is done executing. It indicates that the value in the result register is valid (specific to the meaning of the thread).

• IDLING (0001 0000): If the system issues an IDLE command, the status transitions to IDLING. HWT will return to its previous status, either RUNNING or BLOCKED, after a RUN command.

The argument register may be read from at any time without side effect. Writing to this register has no effect.

**argument**

**Overview**

When a thread is created by the system, the system may pass one argument into the thread. The argument register is used to allow the system to pass in the argument. If used, the system must set the argument after setting the thread_id register and prior to issuing the RUN command.

The meaning of the value of the argument register is thread specific. Generally it is an address pointer to data the thread is to operate on. Setting the argument register is not required.

**Protocol**

The system may write to the argument register only if the status register is USED. This means that the system, when it wants to start the HWT must first issue a RESET command, set the thread_id register, set the argument register (if used), and then issue a RUN command.

Upon a RUN command, the HWTI stores the argument register into the user_result register. It does this prior to changing the user_status from RESET to RUN. The HWTI will maintain this value in the user_result register until the HWTUL issues its first non-NOOP opcode.

**result**

**Overview**

When a thread is created, runs, and then exits, the thread has the option of passing results back to the parent thread. As a note, this is only for joinable threads, results have no meaning in detached threads. To pass back results to the parent, the HWT places the value in the result register. For consistency with the pthreads interface, the result value should be a pointer.

The system may read the result register at any time, although, it only has meaning when the status register reads EXITED.

**Protocol**

When the HWTUL stops, and is ready to issue an EXIT command in the user_opcode register, the HWTUL must place any results it wants to report in the user_argument_data register prior to issuing the EXIT opcode. Once the HWTI sees the EXIT command, it will copy the value of the user_argument_address register into the result register.
The system may read from this register at anytime without side effect. Writing to this register has no effect.

**User Logic Application Programming Interface**

Each of the registers in the User Logic Application Programming Interface (the interface between the HWTI and HWTUL layers) may only be accessed by the HWTUL later. They system layer has no direct access to their values.

The width (number of bits) of each register is given in the Protocol section.

**user_status**

*Overview*

It is used by the HWTI layer to give instructions to the HWTUL layer, as well as provide a handshake mechanism when the HWTUL layer issues commands to the user_opcode register. There are four possible values to the user_status register, they are RESET, RUN, ACK, PAUSE.

*Protocol*

This register is 4 bits. Each bit represents a status. The status, their binary value, and their meaning are as follows:

- **RESET (0001):** This is the initial value of user_status on power up. HWTI will also change to this state if the system issues a RESET to the command register. HWTI will remain in this state until the system issues a RUN to the command register, at which time HWTI will change the user_status register to RUN.

- **RUN (0010):** This value has two meanings, depending on the previous status. If the previous status was RESET, RUN tells the HWTUL to start executing from the top of its state machine. If the previous status was ACK, it signifies to the HWTUL layer that the requesting opcode is complete, and any result is stored in the user_result register.

- **ACK (0100):** If the HWTUL layer issues a command to the user_opcode register, the HWTI layer responds by changing the user_status to ACK. The HWTI will also read the user_argument registers for any data and addresses it is to work with. At this point, the operation has started, but not yet complete, the values in the user_result register are not valid. The HWTUL layer will know that the command is complete, and the user_result value valid, when the user_status register changes back to RUN.

- **PAUSE (1000):** If the system issues an IDLE to the command register, the user_status register changes to PAUSE. It signifies to the HWTUL layer to remain in its current state, until the status register changes back to RUN or ACK (the system is prohibited from issuing an IDLE while the HWT is not running). The HWTI later will always return the user_status register to the status it was prior to changing it to PAUSE. The HWTUL should not change its internal state while the user_status is PAUSE.

Writing to this register has no effect. Reading from this register has no side effect.
user_argument_one / user_argument_two

Overview
Some of the operation codes the HWTUL layer may issue to the HWTI layer require additional information to complete. The user_argument_one, and two registers allow for the passing of this data. They are writable by the HWTUL layer when the user_status is RUN. Writing to these registers is prohibited at other times. This means that the arguments must be set prior to the HWTUL layer issuing a command to the user_opcode register.

In general, knowing what argument to place in which register, is equivalent to knowing the parameter order for the equivalent hthread call. For example, hthread_join(hthread_t, void**), takes two arguments (note that hthread_join is not currently supported). The hthread_t argument is placed in the user_argument_one register. The void** argument is placed in the user_argument_two register.

It is expected, as the HWTI grows to support more hthread system calls, additional registers will be needed. These future registers will be named user_argument_three through six.

Protocol
The user_argument_data register is 32 bits wide.

The HWTUL layer may write to this register when the user_status is RUN.

The HWTI layer will read the address, and take appropriate action, on the opcodes READ, WRITE, EXIT.

The HWTUL layer may read from this register to view the last value is stored in it. Reading this register has no side effect.

user_opcode

Overview
When the HWT is RUNNING, the HWTUL layer may request services via the HWTI layer. The user_opcode register is the mechanism allowing the HWTUL layer to request these services.

The goal is to provide all services to a hardware thread that a software thread has. These services will be performed via the user_opcode. The Protocol section lists the services currently supported. A NOOP operation was added to assist in the handshaking protocol to request a service.

Protocol
The user_opcode register is 8 bits.

Only one user_opcode may be issued at a time.

The HWTUL layer may issue a new opcode (request a service) only when the user_status register reads RUN. The one exception is when the HWTUL layer changes the opcode to NOOP which can only be done when the user_status is ACK. At all other times the opcode is ignored.
The HWTUL layer must maintain the value of the opcode (service request), and any user_argument registers, until the user_status value changes from RUN to ACK. Once the status is changed to ACK, the HWTUL must change the opcode to NOOP, the user_argument registers do not have to be changed. The HWTI layer will indicate that the operation is complete by changing the user_status back to RUN. At this time, any value in the user_result register is valid.

The opcodes, their binary value, and their implementation meaning are as follows:

- **NOOP (0000 0000):** The HWTUL is not requesting any service from the HWTI layer.
- **HTHREAD_EXIT (0000 0001):** The HWTUL has finished executing and is requesting to call exit_thread (on the thread manager), any results are placed in user_argument_address.
- **READ (0000 0010):** The HWTUL is requesting to read the value of the memory addressed in user_argument_address.
- **WRITE (0000 0011):** The HWTUL is requesting to write the data in user_argument_data to the address user_argument_address.
- **HTHREAD_SELF (0000 0100):** Returns the value of the thread_id register to the user_result register.
- **HTHREAD_YIELD (0000 0101):** The hthread_yield call normally allows a CPU bound thread to give up the CPU. Since the HWT is not running on the CPU, the hardware implementation immediately ACKs and returns to RUN. This call is implemented for completeness only.
- **HTHREAD_MUTEX_LOCK (0000 0110):** The HWTUL is requesting to lock the mutex indicated in user_argument_data.
- **HTHREAD_MUTEX_UNLOCK (0000 0111):** The HWTUL is requesting to unlock the mutex indicated in user_argument_data.
- **HTHREAD_INTRASSOC (00001000):** Allows the HWT to associated with an interrupt specified in the user_argument_data register.

### user_result

**Overview**

This register has two uses, first when the HWT is created, and initial arguments are passed in, and second, when the HWTUL layer requests certain services, and the HWTI layer passes values back.

When the HWTUL layer makes a service request, if the particular operation returns a value (READ for example), the result is stored in this register prior to the user_status changing back to RUN.

**Protocol**

The user_result register is 32 bits.
The HWTI layer will place the value of the service request (opcode) in user_request, and then transition from ACK to RUN. The HWTI layer will keep this value constant until the next opcode request.

The HWTUL layer may read this register without side effects. Writing to this register is not permitted.