This section covers inter thread communication

MT Inter Thread Communication

- ITC
  - The ITC block is a piece of logic:
    - Outside of the core and
    - Connects through the gating storage interface.
    - SoC integrators are free to use the MIPS-supplied example logic in whole, in part, or to write their own.
    - This section only describes the programming features of the sample ITC block supplied in the core package by MIPS.

MIPS

2

Inter thread communication is handled by an external logic Block

+ It is connected outside of the core

+ this block connects to the core through the getting storage interface

+ you can design your own ITC block to do exactly what you want it to do

+ But This section deals with an ITC block that has been designed by MIPS as part of your core package, that you are free to use.

MIPS ITC Implementation, Data

- **The reference ITC supports two kinds of storage in a cell:**
  - FIFO queues containing 4 data entries each 32 bits wide
  - Semaphore cells containing 1 data entry that is 32 bits wide
- **All cells should be accessed as 32 bit memory. Accesses such as LH or SB instructions should not be used.**

MIPS

The ITC block is grouped into elements called cells. Two kinds of cell are supported:

+ A FIFO Cell that contains 4 32 bit entries

+ And a semaphore that contains one entry

+ all cells can only be accessed by whole words, in other words you can only do 32 bit loads or stores to a cell location.

MIPS ITC Implementation, Tags

- **Each ITC cell contains a tag entry and 1 or more data entries**
  - **The tag of each ITC cell contains a number of control bits that regulate accesses to that cell:**

Fields		ITC Cell Tag	Read/Write	Reset State
Name	Bits			
FIFODepth	31-28	0 for a single entry Semaphore cell or 2 for four entry FIFO cells.	R	UD
FIFOPtr	20-18	Number of entries to read until cell is empty, reads zero for single entry Semaphore cells	R	UD
FIFO	17	Tells you the type of cell it is. 1 for FIFO cells and 0 for single entry Semaphore cells.	R	UD
T	16	Trap Bit. When set, this bit causes the processor to take a Gating Storage Exception on PV or EF accesses. (Could be used by OS to reused TC).	R/W	UD
F	1	Full Bit. This bit indicates that the cell is full	R/W	UD
E	0	Empty Bit. This bit indicates that the cell is empty. Writing 1 to this bit also reset FIFOPtr.	R/W	TD

MIPS

4

Each cell contains a tag entry that tells you about the cell and allows you some control of the cell. The Tag entry is accessed through the control view that I will go over later.

+ The FIFO Depth field will be set to 0 if this is a semaphore cell or a 2 if this is 4 entry FIFO cell.

+ the PTR field will always be set to 0 if this is a semaphore. For a FIFO cell it indicates the number of cells to read before the cell becomes empty.

+ the FIFO Field will be set to 0 for a semaphore cell and a 1 if the cell is a FIFO.

+ The trap bit will cause a getting storage exception if the thread accessed the cell through a PV or EF view. This can be used by a OS to swap this thread out for another.

+ the F bit will indicate that the cell is full

+ the E bit indicates that the cell is empty. Software can set this bit making the cell empty which will also reset the cell pointer so this is a quick way of clearing a Cell.

MIPS ITC Implementation, Views

- **A View controls the behavior of a access to Cells.**
  - The view is encoded in bits 6:3 of the memory address being accessed.

Address Bits 3 - 6	Cell Address offset	View
0x0	0	Bypass View
0x1	8	Control View
0x2	16	Empty/Full Synchronized View
0x3	24	Empty/Full Try View
0x4	32	P/V Synchronized View
0x5	40	P/V Try View

MIPS

5

A view is the method the thread what's to use to access the cell. The ITC controller will look at bits 3 through 6 of the address being accessed and use them to figure out how the thread wants to access the cell.

The next several slides will go into each view in detail.

MIPS ITC Bypass View

- **The Bypass View Can be used to load or store data to a cell without any other effect.**
  - A load or store does not cause the issuing thread to block and does not affect any of the cells' state bits.
  - A Bypass view store to a FIFO ITC location overwrites the newest FIFO entry, while a Bypass view load returns the contents of the oldest entry.
  - Use this view to set the initial value for a cell.

MIPS 6

The Bypass view is used to bypass the normal operations of the cell.

+ Loads or stores do not cause the issuing thread to block and the cells state bits stored in the cells tags do not change.

+ for a FIFO type of sell a Bypass store overwrites the newest FIFO entry and a load returns the oldest entry.

+ you would this view to set the initial data values for a cell.

MIPS ITC Control View

- **The Control View of the ITC location can be used to manipulate the tag of the ITC cell without any other effect.**
  - All tag loads and stores, access the entire 32-bit tag value.
  - Accesses using Control view never cause the issuing thread to block and never result in Gating Storage exceptions.
  - A Control view store to a FIFO location with the E bit set will cause the FIFO to reset its read pointer (FIFOPtr).

MIPS 7

The control view is use to change the cell's tag values without any other effect.

+ All tag loads or stores will write the entire 32 bit tag value

+ A load or store to a tag will not cause the issuing thread to block or cause an Gating Storage exception

+ If you use the control view to set the empty bit The FIFO pointer also gets reset thus clearing the FIFO at the same time so you don't have to read out all the FIFO locations.

MIPS ITC I Empty/Full Synchronized View

- **The Empty/Full Synchronized View of the ITC location implies that a load causes the issuing thread to block if the cell is Empty. Similarly, a store blocks if the cell is full.**
  - Accesses using this view cause an atomic update of the Empty and Full bits to reflect the new state of the cell.
  - The operation of SC using this view is undefined.
  - If the T bit is set, then any accesses will cause a gated exception trap.
    - T could be set by an OS which wants to keep track of reads and writes, perhaps because it's recycled a TC which was waiting here and wants to know when it might have been unblocked or wants to recycle a TC that will become blocked.

MIPS 8

The Empty Full Synchronized View is used to write 32 bit data to the Cell. If the cell is a FIFO the data is read in a First in first out manner. Reading an empty cell will cause the executing thread to block until something is written by another thread to the cell. Writing to a full cell will block until the cell is read by another thread.

+ The Empty Full bits are evaluated and update appropriately with each load or store to the cell.

+ You should not use the Store Conditional instruction with this view.

+ If the Trap bit is set any access will cause a gating storage exception. This can be used by an OS to reuse threads that would be idle waiting something to get written to the cell. The OS would have to save the context of the Thread then it could use the thread for another process. When new data is written to the cell another exception would happen and the OS would know that it need to find a thread to continue processing.

MIPS ITC Empty/Full Try View

- **The Empty/Full Try View of the ITC location is similar in nature to the previous E/F Synchronized but it does not block if an access fails. It differs in the following ways:**
  - A load returns a value of zero if the cell is empty.
  - A store (SW) instruction to Full locations fail silently.
  - Store Conditional (SC) instructions will indicate success or failure based on whether the ITC store succeeds or fails.

MIPS

9

The Empty Full Try view is simulator to the previous Empty Full Synchronize view but it is intended to work with the Load Link Store conditional instructions.

+ instead of blocking a load will return a 0 if the cell is empty.

+ a store to a full location will not block just fail silently

+ but a Store Conditional instruction can be use to tell you if the store succeeded or not.

MIPS ITC P/V Synchronized Views

- **The P/V Synchronized View implements a "P/V" counting semaphore, "p" and "v" are the "wait if zero, then count down" and "count up" functions respectively.**
  - A load from a zero cell, blocks until a non-zero value appears. Otherwise the load returns the value and atomically decrements the stored value.
  - Any store causes an atomic increment of the cell value, up to a maximum value of 2^{16-1}, at which it saturates.
  - P/V operations do not modify the empty and full bits, which should both be cleared before an entry is used for P/V purposes.
  - The P/V view of a FIFO Cell doesn't make sense, and the result of any such access is undefined.

MIPS

10

The P/V Synchronized View implements a P/V counting Semaphore

+ a load will block if the semaphore value is 0 or decrement the value and return the old value.

+ a store does not actually store a value to the semaphore instead it increments the value in the semaphore.

+ The Empty/full bits have no meaning when using this view and are not updated.

+ Don't use this view with a FIFO cell.

If the trap bit is set in the Cell's tag then any access will cause a gating storage exception as discussed in previous slides.

MIPS ITC P/V Try View

- **The P/V Try View is similar in nature to the previous P/V Synchronized except it does not block if the access fails. It differs in the following ways:**
  - A load with this view returns a value of zero even if the cell contains a data value of zero.
  - The operation of Store Conditional instruction using this view is undefined.

MIPS

11

The Try version of the P/V view never blocks.

+ If the cell value is 0 a load will return a 0

+ don't used the Store Conditional instruction with this view

If the trap bit is set in the Cell's tag then any access will cause a gating storage exception as discussed in previous slides.

MIPS ITC Block Configuration

- **The configuration information for the ITC space is held in two "tags".**
- **Access to these tags is done by**
  - Set the *ErrCtl[ITC]* bit to access ITC space configuration "tags"

```
errctlreg = mips32_geterrctl();
errctlreg_withITC = errctlreg | ERRCTL_ITC;
mips32_seterrctl(errctlreg_withITC);
```

  - cache Index_Load_Tag_D instruction
  - cache Index_Store_Tag_D instruction

MIPS

12

The ITC Block contains 2 tags that can be used to read the current configuration and configure the ITC block.

+ To access the ITC block you need to set the ITC bit in the CP0 Error Control register. This will direct the cache instruction to the ITC Block instead of the cache block.

+ Here is an example of the C code needed to set the ITC bit.

+ to read a tag use the cache index load tag data instruction

+ to write a tag use the index store tag data instruction

There is an example at the end of this section that will show the instructions need to read and write these tags

MIPS ITC Configuration Tag (address 0)

Fields		Address Map Register Tag0 Offset 0	Read/Write	Reset State
Name	Bits			
BaseAddress	31-10	ITC Base Address (Physical)	R/W	UD
ITC_En	0	ITC Enable – clear at reset (hides ITC) setting it to 1 enables ITC block	R/W	UD

MIPS

13

The first ITC tag is located at offset 0 it contains the Base address of the ITC Block and the ITC Enable bit.

+ The base address should be set when you initialize the ITC block. The address uses the top 22 bit of this register along with the address mask, that I'll explain on the next slide, to determine the starting address of the ITC block.

+ the ITC_En bit enables the ITC block. This should be set when you want to start using the block.

NOTE: If you position the ITC block over real memory or device I/O memory you will no longer be able to access that memory or device.

MIPS ITC Configuration Tag (address 8)

Fields		Address Map Register Tag1 Offset 8		Read/ Write	Reset State
Name	Bits				
Num Entries	30-20	Number of ITC cells present		R	Preset
AddrMask	16-19	Indicates which bits of the BaseAddress field should not participate in determining an ITC memory hit. This field effectively defines the size of the ITC memory block. AddrMask set to zero implies a 1KB ITC address space, and AddrMask set to 0x3f implies a 128KB address space.		R/W	UD
Entry Grain	2-0	Interval spacing between ITC Cells		R/W	UD
		Encoding	Size in Bytes		
		0x0	128		
		0x1	256		
		0x2	512		
		0x3	1024		
		0x4	2048		
		0x5	4096		
		0x6	8192		
		0x7	16384		

MIPS

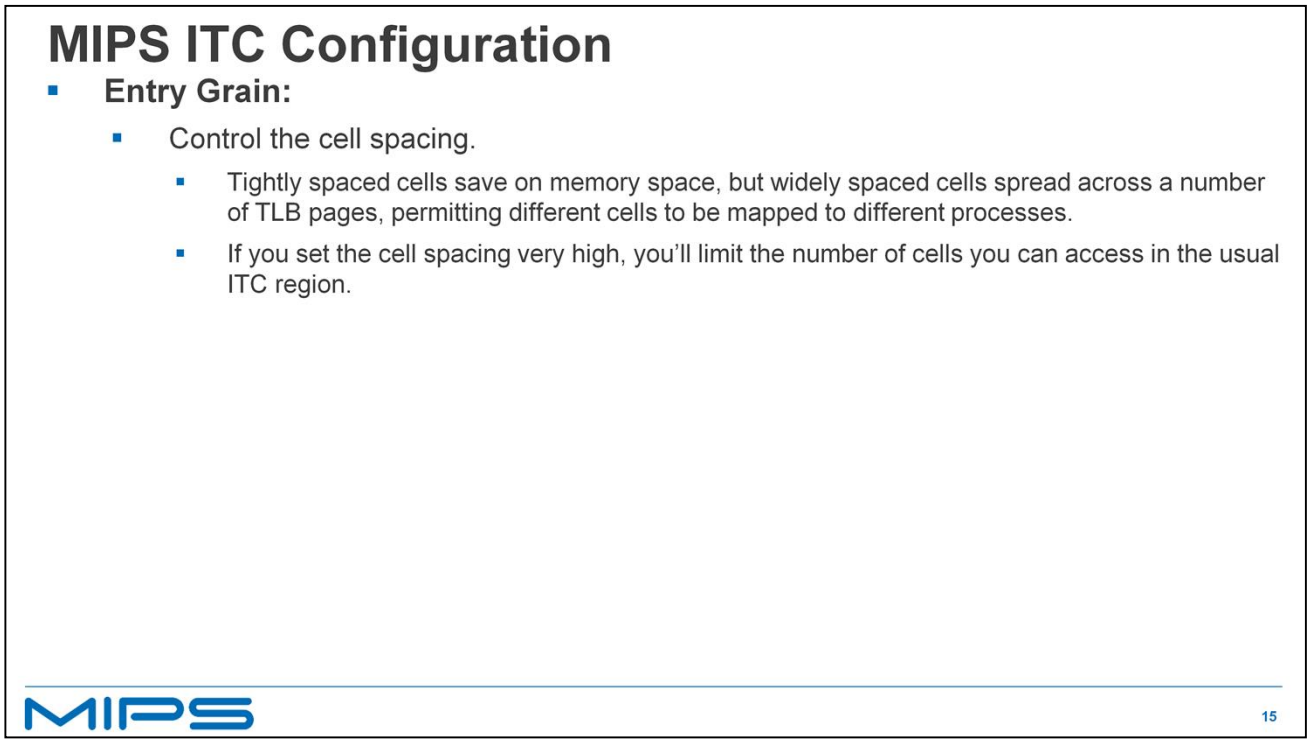
14

The second ITC tag is located at offset 8 and contains the number of ITC entries you have, the Address mask and Cell Entry Grain.

+ the Number of ITC cells is static for you core and this field tells you how many you have.

+ you can set the address mask field. This is an address mask. It Indicates which bits of the BaseAddress field should not participate in determining an ITC memory hit. This field effectively defines the size of the ITC memory block. AddrMask set to zero implies a 1KB ITC address space, and AddrMask set to 0x3f implies a 128KB address space.

+ You can set the Entry Grain to configure the interval spacing of the ITC Cells This chart gives you the Entry value and the corresponding size of the interval.



Here are some ideas of what you would do about the Entry Grain value

+ For simple semaphore cells that are used to communicate between processes and the OS you will probably want to space them a tightly as possible this will save on memory map space.

For FIFO cells that will be used to hold data within a process that will not be shared by other processes you can space the cells wider apart on different pages so the cells can be mapped to the process and protected buy the TLB.

One thing to take into account is,

Depending on the setting of the *AddrMask, NumEntries, and EntryGrain, it is possible that ITC cells do not fill up the*

entire ITC address block. If for example, if you have only two cells and they are mapped to a 1KB area with a stride of 256B that is the *EntryGrain equal* to 0x1, the first cell starts at offset 0x000 and the second at offset 0x100. The remaining two 256B regions starting at offsets 0x200 and 0x300 do not map to any storage. Any access to an address that does not map to an ITC entry will result in undefined behavior.

+ It is also possible to set the cell grain entry too large for the address mask which would make some of the cells unavailable.

MIPS ITC Configuration

- **Once this is set up and enabled (ITC_En),**
  - All accesses to this physical address range will go to ITC, and will no longer show up on the main system interface.
  - These locations will “overlay” anything else you expected to be there. Take care not to overlap any vital address.
  - Don’t forget to clear *ErrCtl[ITC]* afterwards, so that cache operations can continue as usual.

MIPS

16

Once the ITC_Enable bit is set in the ITC tag.

+ All access for the memory region will go to the ITC controller

+ Anything that was in that address range will no longer be accessible

+ After you are finished initializing the ITC Block make sure to clear the ITC bit in the ErrCtl register so cache instructions will once again be directed to the Cache controller.

MIPS ITC Example

- **ITC Example**
  - 4 TCs running on 1VPE.
  - Uses P/V view.
  - Each TC will block all others while it increments a counter to 2000. Then it will release the semaphore and the next TC will enter the code.

I am now going to tie this al together by going through a code walk through of the use of the ITC block as a semaphore.

+ In the example I will be using 4 threads on 1 VPE

+ I will use the P/V view for the semaphore.

+ In this simple program each thread will acquire a semaphore count to 2000 and then release the semaphore so another thread can count

Configuring the ITC tags

- **The code is similar to the other examples with the following major changes.**
  - Configure the ITC tags. This is done by using cache opts. To set the cache mode for ITC tags the ITC bit in the ErrCtl register must be set.

```
errctlreg = mips32_geterrctl();
errctlreg_withITC = errctlreg | ERRCTL_ITC;
mips32_seterrctl(errctlreg_withITC);
```

MIPS

18

I will go over the code changes made to the basic thread example I show you earlier.

+ First I am going to configure the ITC Block. To get to the ITC block's controller to configure its tags you need to turn on the ITC bit in the Error Control register. Once you do that you can use the cache instruction to read or write to the ITC block's tags.

Here is the sequence of C instruction to turn on the ITC configuration bit.

+ As you can see I use the mips32_geterrctl macro to get the register

+ then turn on the ITC bit using the ERRCTL_ITC #define

+ and last write the Error Control with the new value using the mips32_seterrctl macro

Configuring the Address mask bits and Entry Grain

- **Code Changes continued:**
  - Configure the Address mask bits and Entry Grain.

```
ITC_Config_Tag8 = ( (ITC_AddrMask << 9) |  ITC_EntryGrain);
mips32_setdtaglo(ITC_Config_Tag8);
```

MIPS

19

Now I am set to configure the ITC's address mask and Entry grain.

Since I am going to use the cache instruction to write the ITC's tags I must first setup the Data Tag lo register with the correct information. The cache instruction will write the value in the data tag low register to the ITC tag.

+ here is the code to set the address mask and the entry grain using #define values for the address mask and entry grain.

+ this code moves the value to the data tag low register using the mips32_setdtaglo macro

Cache Opt, “index store tag” for Tag 1

- **Code Changes continued:**
  - Cache Opt, “index store tag” to set ITC Tag 1 (offset 8)

```
__asm__ volatile
  ("\
  cache 9, 8($0);  \
  ehb; \
  "
  ) ;
```

MIPS

20

Now that the Data tag lo register is set up I can use the cache instruction to write it to the ITC’s tag register.

+ I’ll use inline assemble code to do that.

+You can see the cache instruction takes 2 arguments:

+ the first is the command operation you want it to perform which is operation 9, index store tag

+ the second argument is, which location within the ITC tag memory you want to write. In this case I am using GPR 0 with an offset of 8 bytes to write ITC tag 1.

Configure the Base Address and ITC_En

- **Code Changes continued:**
  - Configure Base address and ITC_En (enable bit) in ITC tag index 0 and
    - Use physical base address

```
ITC_BlockNC = (unsigned int *)
                ((unsigned int)ITC_Block & 0x7fffffff);
```

    - Set tag

```
ITC_Config_Tag0 = ((unsigned int)
                        ITC_BlockNC | ITC_En);
mips32_setdtaglo(ITC_Config_Tag0);
```

Fields		Address Map Register Tag0 Offset 0	Read/Write	Reset State
Name	Bits			
BaseAddress	31-10	ITC Base Address (Physical)	R/W	UD
ITC_En	0	ITC Enable – clear at reset (hides ITC) setting it to 1 enables ITC block	R/W	UD

MIPS

21

Now I'm going to configure the ITC tag 0 with the Base address of the ITC block and enable the ITC Block.

+ When I reserved space for the ITC block I just declared an array. I did this to make sure no other variables would be placed in that memory mapped range. I could have just selected a physical address that is not mapped to any device. Because I declared the ITC as an array in C it is given a virtual address in Kseg 0. Since addresses in Kseg 0 are mapped to physical address starting a 0 all I need to do convert this virtual address to a physical address is strip off the top bit.

+ next I or in the enable bit

+ then write the value to the data tag lo register using the mips32_setdtaglo macro.

Cache Opt, “index store tag” for Tag 0

- **Code Changes continued:**
  - Cache Opt, “index store tag” to set ITC Tag 0

```
__asm__ volatile
  ("\
  cache 9, 0($0); \
  ehb; \
  "
  ) ;
```

MIPS

22

Now I use the cache instruction to write the Data Tag lo register to the ITC tag 0 the same as I did for ITC tag 1

+ Except this time I use a offset value of 0 for Tag 0

Enable ITC Entry

- **Code Changes continued**
  - Return ErrCtl to its previous state

    mips32_seterrctl(errctlreg);
  - Then Enable ITC Entry
    - Use cached address

      (from 0x80000000 to 0xA0000000)

    ITC_BlockNC = (unsigned int *)

    ((unsigned int)ITC_Block | 0x20000000);

MIPS

23

Now that I am done with writing the ITC Block Tags I will turn off the ITC bit in the Error Control register so the cache instruction operation will refer to the cache controller instead of the ITC controller.

+ So far I have enabled the ITC Block but not the actual ITC Cell I will be using. To do this I need to use an uncached address to access the Cell. I can do this by changing the address of the Cell array from being in KSEG 0 to KSEG 1. All I need to do is change the top bit of the address.

+ here is the C code to do that.

Enable ITC Entry

- Use control view to access Entry Tag

  ITC_Cell =(unsigned int *)((unsigned int)ITC_BlockNC | ITC_ControlView);

- Write Tag

  *ITC_Cell = ITC_E;

Fields		ITC Cell Tag	Read/Write	Reset State
Name	Bits			
FIFODepth	31-28	0 for a single entry Semaphore cell or 2 for four entry FIFO cells.	R	UD
FIFOPtr	20-18	Number of entries to read until cell is empty, reads zero for single entry Semaphore cells	R	UD
FIFO	17	Tells you the type of cell it is. 1 for FIFO cells and 0 for single entry Semaphore cells.	R	UD
T	16	Trap Bit. When set, this bit causes the processor to take a Gating Storage Exception on PV or EF accesses. (Could be used by OS to reused TC).	R/W	UD
F	1	Full Bit. This bit indicates that the cell is full	R/W	UD
E	0	Empty Bit. This bit indicates that the cell is empty. Writing 1 to this bit also reset FIFOPtr.	R/W	TD

MIPS

24

Now that I have the correct address I need to set the lower bits of the address to tell the ITC controller which view I what to use. To enable the Cell I need to set the enable bit and to do that I need to use the Control View so I set those bits.

+ Then I write the enable bit to the Cell.

Increment ITC Cell value

- **Code Changes continued**
  - Next, Increment ITC cell using PV view so first access will not block.

```
ITC_Cell =(unsigned int *)((unsigned int)ITC_BlockNC | ITC_PVSyncView);
*ITC_Cell = 1;
```

MIPS

Since I don't want the semaphore to block I want the cell to start out with a 1.

+ To do this I can just use the PV Sync view where any writes to the Cell will increment its contents. So I or in the P/V Sync View Bits Using the #define ITC_PVSyncView and write to the address.

Using the Semaphore

- **Code Changes continued**
  - In the count function:
    - Using PV view read ITC entry. This will block if entry is 0.

```
ITC_Cell =(unsigned int *)
            ((unsigned int)ITC_BlockNC | ITC_PVSyncView);
ITC = *ITC_Cell;
```

MIPS

In the count function before we enter the counting loop I what to add the code so not more than one thread will be counting at a time.

This is where I will read the Cell using the PV Sync view.

+ To do this I or in the PV Sync View bits into the Cell address and then read the Cell

Since I initially incremented the cell the first thread to get to this point will get the semaphore and decrement the Cell counter to 0. The next thread that reads the Cell will block waiting until the Cell counter in incremented before it can continue.

MIPS ITC Example

- **Code Changes continued**
  - After counting is finished release semaphore:
    - Use P/V view to write back read in value.

```
*ITC_Cell = ITC;
```

MIPS

27

I then add code after the count loop to write to the Cell. This will increment the Cell to 1 so that another thread can continue.