Comparator: New Design

Note: Changes were made in the module used to compute the fingerprints. In particular, the block size variable was removed so that all store instructions get compressed into a single fingerprint. Due to a loss of code, the original changes were lost. This is currently being implemented by setting the *carry* signal (Line 43, **counter.v**) to 0. While the goal is achieved, there is a great deal of unnecessary hardware in the fingerprint unit that should be dealt with.

Top level

The top level hardware block that performs the comparison is the **CFPU**. It has three bus interfaces that are described in Table 1:

Table 1 - CFPU bus interfaces

Name	Type	Description
csr	Avalon Slave	• Program internal registers of the **CFPU** through writes • Provide task success and failure information through reads
fprint	Avalon Slave	• Takes task start and finish strobes and fingerprints from all the secondary cores through writes. No read interface
checkpoint	Avalon Master	• Sends interrupts to cores working on a same task to notify them when all other cores on the task have reached the same checkpoint location

The CFPU consists of 4 submodules shown in Figure 1.

- **comparator**
- **checkpoint**
- **fprint_registers**
- **csr_registers**

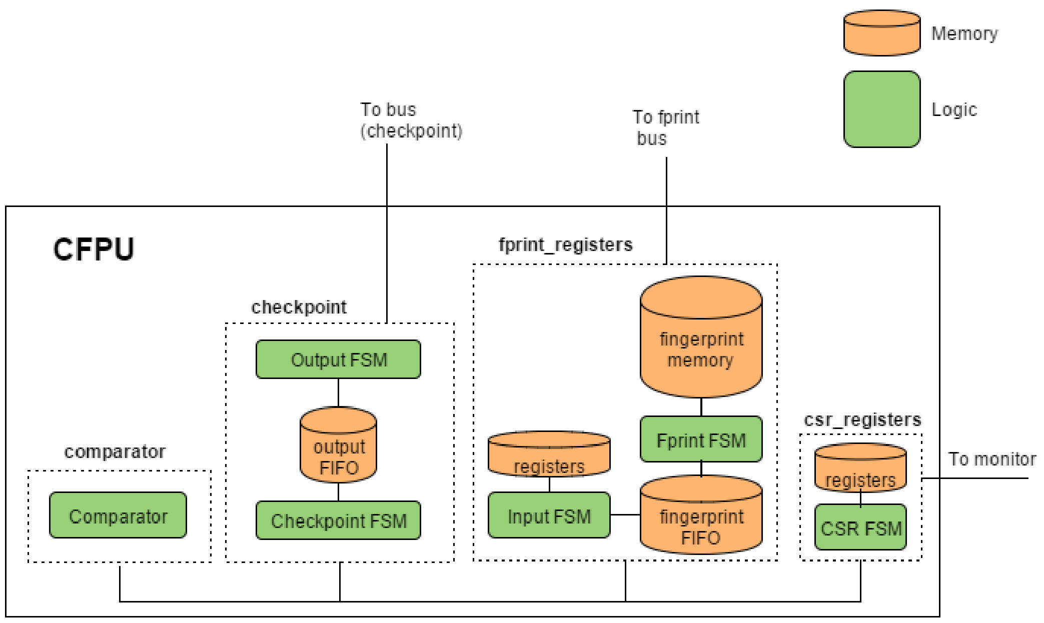

Figure 1 - Block level diagram of the CFPU

csr_registers

This module controls the **csr** bus interface. It also contains all the programmable registers of the **CFPU**, and relays the information to the other submodules via internal signals.

The description of the **csr** bus signals regarding the registers that can be accessed are listed in Table 2.

Table 2 - Registers in csr_registers

Name	Type	Address	Databits	Description
Core Allocation Table	Write	0x5000010	writedata(25:24) = Logical ID writedata(19:16) = Task ID writedata(3:0) = Core ID	This is a 16x3 slot SRAM of 4bits in each space. Indexed by Task ID and Logical ID, and the memory content is the core ID

Task Success Count Register	Write	0x5000018	writedata(19:16) = Task ID writedata(9:0) = Success Count	This is a 16 slot SRAM, indexed by Task ID, and the memory content is the successful task completion count.
NMR register	Write	0x5000014	writedata(19:16) = Task ID writedata(0) = NMR info	This is a 16 bit register indexed by Task ID. The corresponding bit is set high when TMR is activated for the task
Exception Register	Read/Write	0x5000000	writedata(:)	This register contains the interrupt bit for task completion/failure. The write is to reset the interrupt
Success Register	Read	0x5000008	-	16 bit register. If a task completes successfully, the corresponding bit is set high
Fail Register	Read	0x500000C	-	32 bit register. If a task fails, the corresponding two bits contain the failing Logical ID. Default value is all 1's

This signals between this module and the rest of the modules and their description is listed in Table 3.

Table 3 - Signals from csr_registers to other submodules

Module	Signal and Description
fprint_registers	• *fprint_task_id* – the four bit Task ID which the incoming fingerprint (on the fprint bus) belongs to • *fprint_physical_core_id* – The four bit core id of the core that is sending the fingerprint • *fprint_logical_core_*id – the two bit logical core id from the Core Allocation Table corresponding to the above task id and core id • *fprint_nmr* – one bit wire that is asserted when TMR is active for the Task ID in '*fprint_task_id*'
comparator	• *comparator_status_write* – write signal from **comparator** to indicate task completion or failure • *comparator_task* – the four bit task id that the comparator is writing the status for • *comparator_logical_core_id* – the Logical ID of the failing core • *comparator_mismatch_detected* – this wire is 'high' is a mismatch in fingerprints has been detected • *csr_status ack* – pulse sent by **csr_registers** to acknowledge status write • *comparator_nmr* – one bit wire that is asserted when TMR is active for the Task ID in '*comparator_task_id*' • *csr_task_maxcount* – The programmed task success count
checkpoint	• *checkpoint_logical_core_id* – The two bit Logical ID needed to get the corresponding Physical ID • *checkpoint_physical_core_id* – the four bit Physical core id from the Core Allocation Table corresponding to the above Logical ID and the **comparator** Task ID

fprint_registers

This submodule controls the **fprint** bus interface. All writes on the bus are first stored in an internal FIFO to minimize time spent on the bus and to achieve parallelism.

When a fingerprinting task begins or ends on a core, it must notify the **CFPU**. This is done by means of a checkout and checkin register. They are each a 16x3 slot register, with one bit at each entry, and indexed by Task ID and Logical core ID. When a task begins on a logical core, the corresponding bit in the checkout register is asserted, and when a task completes on a core, the corresponding bit in the checkin register is asserted.

Since only one fingerprint is expected per task, the fingerprint directory SRAM is removed from the previous design and three 16 slot fingerprint SRAM memories are added, one for each logical core. The fingerprint is stored at the index corresponding to the task ID.

The processing of fingerprints and internal registers are handled by the FSM shown in Figure 2, and described in Table 4. TMR is achieved using a logical AND with the NMR bit from **csr_registers** to determine whether signals from the third logical core should be considered or not.

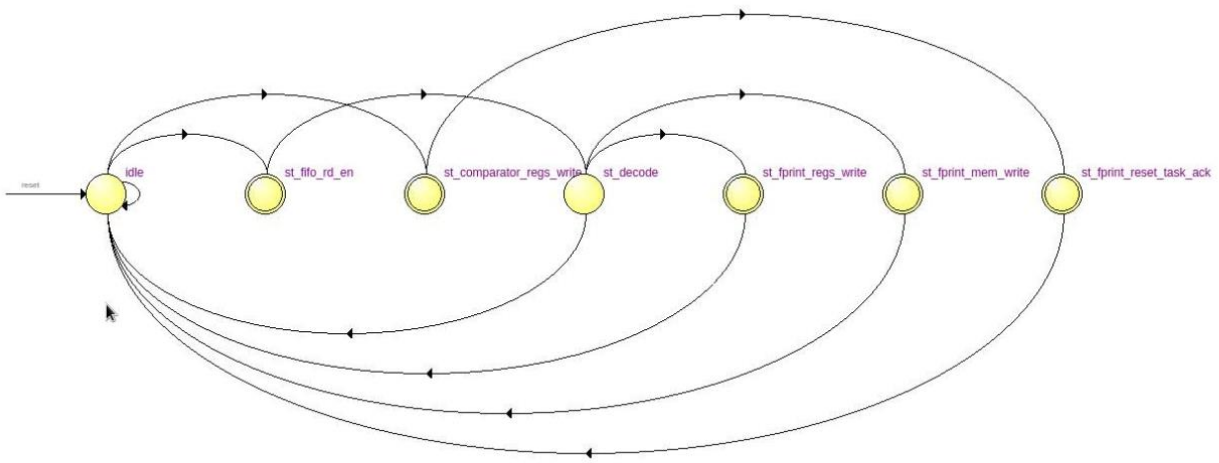

Figure 2 - fprint_registers FSM

Table 4 - fprint_registers FSM description

State	Description
idle	If the comparator sends a reset task signal, go to ‘st_comparator_regs_write’. Otherwise if the **fprint** bus FIFO is not empty, go to ‘st_fifo_rd_en’. Otherwise stay in ‘idle’
st_comparator_regs_write	Reset all the checkin and checkout register bits for both cores corresponding to **comparator** Task ID, go to ‘st_fprint_reset_task_ack’
st_fprint_reset_task_ack	Send an acknowledge signal to **comparator** and return to ‘idle’
st_fifo_rd_en	One clock cycle to get FIFO contents. Go to ‘st_decode’
st_decode	If the write is for the checkout or checkin registers, go to ‘st_fprint_regs_write’, otherwise check if the core sending the fingerprint has checked out. If yes, go to ‘st_fprint_mem_write’, otherwise go to ‘idle’
st_fprint_regs_write	Set the checkout/checkin register, go to ‘idle’
st_fprint_mem_write	Store the fingerprint, go to ‘idle’

checkpoint

This submodule controls the **checkpoint** bus interface. It simply gets a checkpoint signal from **comparator**, and then loops through all the cores for the task and sends them interrupts.

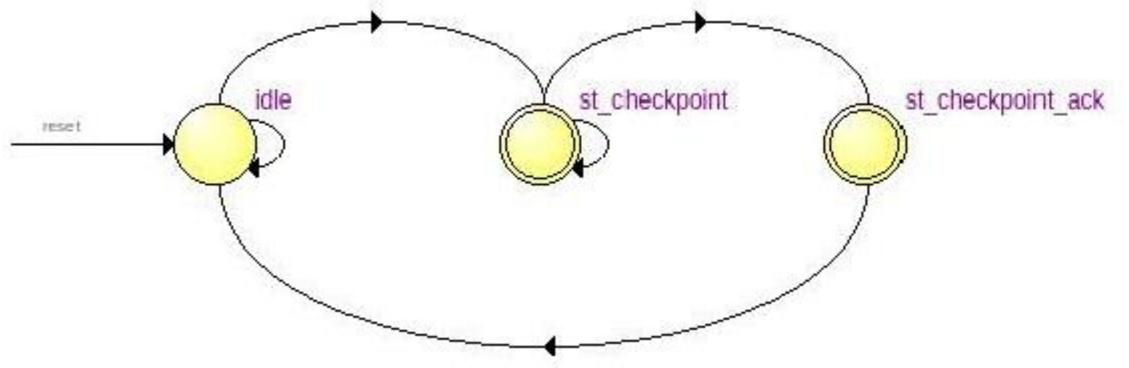

Figure 3 - checkpoint FSM

Table 5 - checkpoint FSM description

State	Description
idle	If **comparator** sends checkpoint signal, go to ‘st_checkpoint’, otherwise stay in ‘idle
st_checkpoint	Wait until all the cores have been sent an interrupt, then go to ‘st_checkpoint_ack’
st_checkpoint_ack	Send an acknowledge signal to **comparator** and return to ‘idle’

comparator

This submodule is responsible for comparing fingerprints and writing the task completion/failure status to **csr_registers**. It receives a checkin signal for all tasks from **fprint_registers** that has the appropriate bit asserted when all cores for the task have checked in. It uses this signal to determine task completion.

This submodule also has the task successful completion counter, which it increments on every task completion and then checks to see if the programmed count has been reached, at which point the status is written to **csr_registers** and the count is reset.

The FSM that implements this submodule is described in Table 6. TMR is achieved using a logical AND with the NMR bit from **csr_registers** to determine whether signals from the third logical core should be considered or not.

Table 6 - comparator FSM description

State	Description
idle	If the checkin signal is high for any task, go to 'st_set_task', otherwise stay in 'idle'
st_set_task	Latch the Task ID of the task that has checked in, go to 'st_load_fprint'
st_load_fprint	One clock cycle to get the fingerprint from **fprint_registers**, go to 'st_compare_fingerprints'
st_compare_fprints	If the fingerprints match, go to 'st_comparator_checkpoint', otherwise go to 'st_comparator mismatch_detected'
st_comparator_checkpoint	Send the checkpoint signal to **checkpoint**, wait for the acknowledge pulse and then go to 'st_task_count_inc'
st_task_count_inc	Increase the task count, go to 'st_fprint_reset_task'
st_comparator_mismastch_detected	Assert and latch the comparator mismatch detected signal (will be reset when the state goes to idle), go to 'st_fprint_reset_task'
st_fprint_reset_task	Send the reset signal to **fprint_registers** and wait for the acknowledge signal. Then if either the task success count is reached or if comparator mismatch detected is asserted, do to 'st_task_count_reset', otherwise go to 'idle'
st_task_count_reset	Reset the task completion count to 0, go to 'st_comparator_status_write'
st_comparator_status_write	Assert the status signals to **csr_registers**. Wait for the acknowledge signal, and then go to 'idle'

Changes to be made/Future Plans

- TMR logic needs modification: Right now, there is no voting taking place and the task fails in TMR mode as soon as any one of the core fails. One way to implement voting is to duplicate the NMR register for all three logical cores. Then TMR will imply all three NMR reg bits are set. Now if there is a failure in a single core, then the NMR bit of the failing core is reset and the task continues with two cores until it completes or another fails.