Comparator with overflow and NMR logic

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
oflow	Avalon Master	• Sends directory overflow and underflow interrupts to each physical core

The CFPU consists of 5 submodules shown in Figure 1.

- **comparator**
- **comp_registers**
- **oflow_registers**
- **fprint_registers**
- **csr_registers**

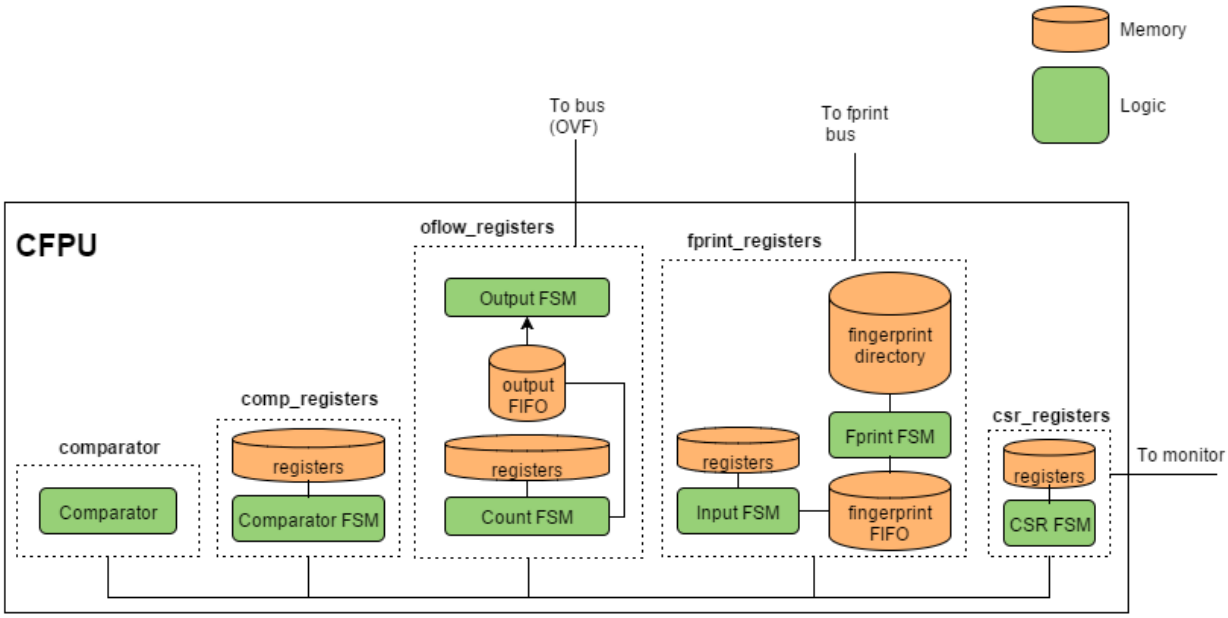

Figure 1 - Block level diagram of the CFPU

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	0x50000D0	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
Directory Start Pointer	Write	0x5000040	writedata(25:24) = Logical ID writedata(19:16) = Task ID writedata(9:0) = Pointer data	This is a 16x3 slot SRAM, indexed by Task ID and Logical ID, and the memory content is the directory start pointer.
Directory End Pointer	Write	0x5000080	writedata(25:24) = Logical ID writedata(19:16) = Task ID writedata(9:0) = Pointer data	This is a 16x3 slot SRAM, indexed by Task ID and Logical ID, and the memory content is the directory end pointer.
Max Count Register	Write	0x50000CC	writedata(25:24) = Logical ID writedata(19:16) = Task ID writedata(9:0) = Max Count	This is a 16x3 slot SRAM, indexed by Task ID and Logical ID, and the memory content is the maximum fingerprint count.

NMR register	Write	0x50000D8	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	0x50000C0	writedata(:)	This register contains the interrupt bit for task completion/failure. The write is to reset the interrupt
Success Register	Read	0x50000C4	-	16 bit register. If a task completes successfully, the corresponding bit is set high
Fail Register	Read	0x50000C8	-	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
comp_registers	• *csr_task_id* – the Task ID for which the pointer is being written • *csr_logical_core_id* – the Logical ID for which the pointer is being written • *csr_start_pointer_write* – a write signal for the start pointer data from **csr_registers** to **comp_registers** • *csr_end_pointer_write* – a write signal for the end pointer data from **csr_registers** to **comp_registers** • *csr_pointer_data* – the start/end pointer data • *comp_pointer_ack* – an acknowledge of the write signal from **comp_registers** to **csr_registers**

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*'
oflow_registers	• *oflow_task_id* – The four bit Task ID of the overflowing/underflowing task • *oflow_logical_core_id* – The two bit Logical ID needed to get the corresponding Physical ID • *oflow_physical_core_ID* – the four bit Physical core id from the Core Allocation Table corresponding to the above Task ID and Logical ID • *csr_fprint_maxcount* – The maxcount value corresponding to the above Task ID and Logical ID • *oflow_nmr* – one bit wire that is asserted when TMR is active for the Task ID in '*oflow_task_id*'

comp_registers

This submodule is in charge of keeping track of the head and tail pointers of the fingerprint directory for each task. It stores the start and end directory locations from **csr_registers** for each Logical core for each task, and includes wrap around logic for both the head and tail pointers corresponding to these values. The functionality is handled by the FSM shown in Figure 2, and described in Table 4.

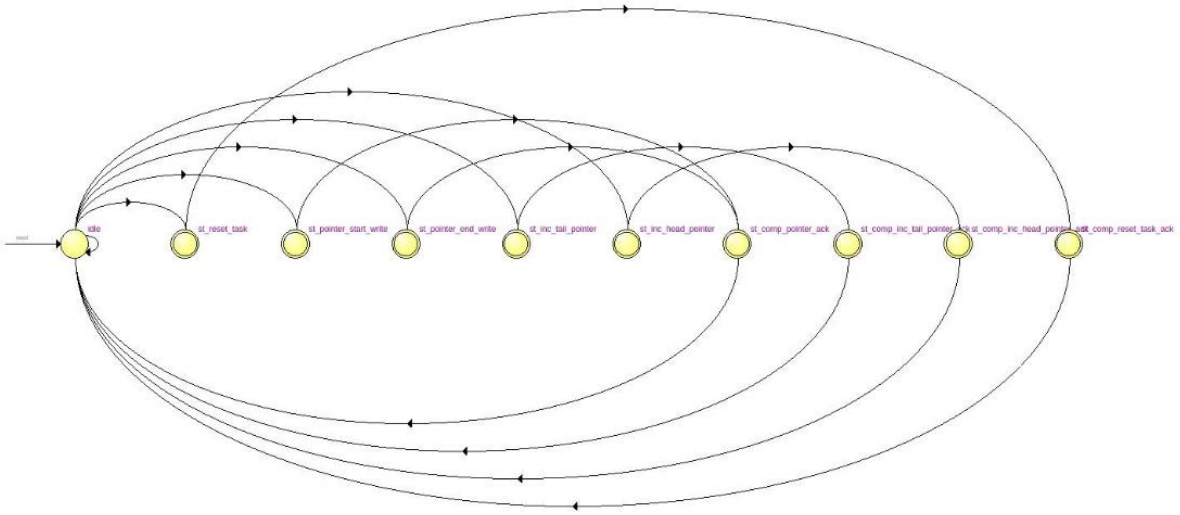

Figure 2 - comp_registers FSM

Table 4 - comp_registers FSM description

State	Description
idle	If **csr_registers** sends a write start pointer signal, go to 'st_pointer_start_write. Else ff **csr_registers** sends a write end pointer signal, go to 'st_pointer_end_write. Else if **fprint_registers** sends an increase head pointer signal, go to 'st_inc_head_pointer'. Else if **comparator** sends an increase tail pointer signal, go to 'st_inc_tail_pointer'. Else if **comparator** sends a reset task signal, go to 'st_reset_task'. Otherwise stay in 'idle'
st_pointer_start_write	Store the start pointer information, go to 'st_comp_pointer_ack'
st_pointer_end_write	Store the end pointer information, go to 'st_comp_pointer_ack'
st_comp_pointer_ack	Send an acknowledge pulse to **csr_registers**, and go to 'idle'
st_inc_head_pointer	Increment the head pointer, go to 'st_comp_inc_head_pointer_ack'
st_comp_inc_head_pointer_ack	Send an acknowledge pulse to **fprint_registers**, and go to 'idle'
st_inc_tail_pointer	Increment the tail pointer, go to 'st_comp_inc_tail_pointer_ack'
st_comp_inc_tail_pointer_ack	Send an acknowledge pulse to **comparator**, and go to 'idle'
st_reset_task	Reset the head and tail pointers to initial values corresponding to the **comparator** Task ID, and go to 'st_comp_reset_task_ack'
st_comp_reset_task_ack	Send an acknowledge pulse to **comparator**, and go to 'idle'

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.

The processing of fingerprints and internal registers are handled by the FSM shown in Figure 3, and described in Table 5. 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.

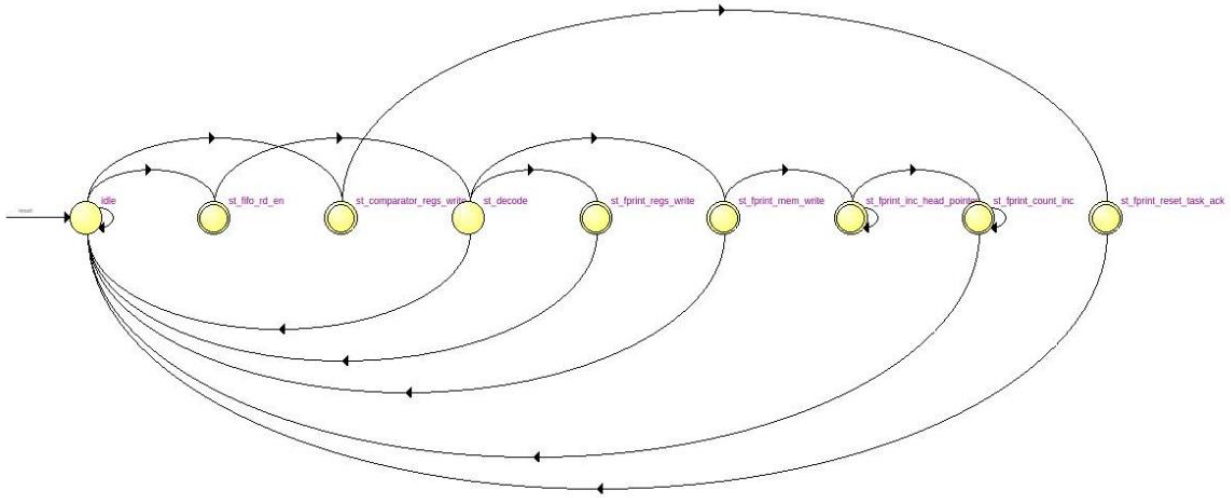

Figure 3 - fprint_registers FSM

Table 5 - 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 at the appropriate location in the directory. Since the fingerprints arrive in two halves, if it is the first half then go to 'idle', but if it is the second half then go to 'st_fprint_inc_head_pointer'
st_fprint_inc_head_pointer	Send a signal to **comp_registers** to increase the directory head pointer. Wait for an acknowledge signal, and then go to 'st_fprint_count_inc'
st_fprint_count_inc	Send a signal to **oflow_registers** to increase the fingerprint count. Wait for an acknowledge signal, and then go to 'idle'

The fingerprints are stored in an internal dual port SRAM directory that is controlled by head pointer and tail pointer signals from **comp_registers**. New fingerprints are written at the location corresponding to the head pointer, and **fprint_regsiters** outputs the fingerprints at the tail pointer location for the comparator to compare.

oflow_registers

This submodule controls the **oflow** bus interface. It keeps track of the fingerprint count for each logical core, and sends an interrupt to the corresponding core when its fingerprint count has exceeded the maximum count as dictated by the programmed value in **csr_ragisters**.

This submodule contains three 16 bit overflow status registers (one for each logical core) that asserts a corresponding bit when a task has overflowed. An *overflow* occurs when a logical core exceeds its max count. At this point, the appropriate bit in the overflow status register is asserted. An *underflow* occurs when the fingerprint count for all cores of an overflowing task (any overflow status reg bit = 1) goes down to 0.

There is an output FIFO to track *overflow* and *underflow* events. When an *overflow* occurs, the Task ID and Core ID are written directly to the FIFO. When an *underflow* occurs, submodule loops through all logical cores, and writes its information to the FIFO if the overflow status bit for that core was asserted.

This submodule also contains fprint ready and fprint remaining registers that are sent to the **comparator**. These signals were handled by **comp_registers** in the previous design, and are now being handled by this submodule due to changes in the directory access structure.

The functionality is achieved by the FSM shown in Figure 4, and 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.

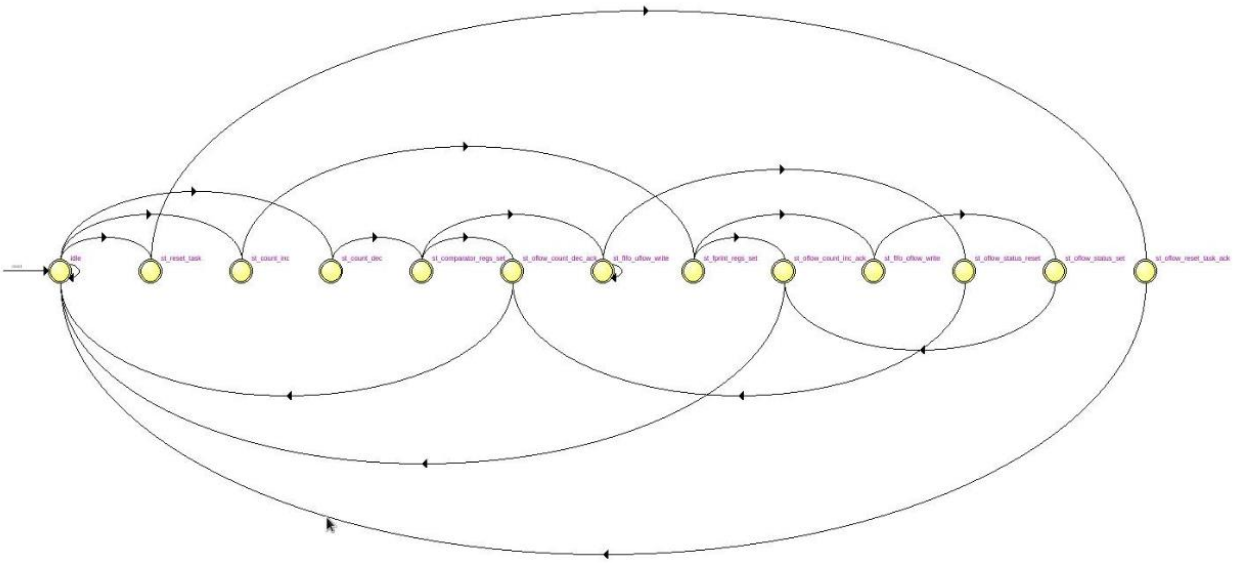

Figure 4 - oflow_registers FSM

Table 6 - oflow_registers FSM description

State	Description
idle	If **comparator** sends a reset task signal, go to 'st_reset_task'. Else if **fprint_registers** sends a count increase signal, go to 'st_count_inc'. Else if **comparator** sends a count decrease signal, go to 'st_count_dec'. Otherwise stay in idle
st_count_inc	Increase the fingerprint count, go to 'st_fprint_regs_set'
st_count_dec	Decrease the fingerprint count, go to 'st_comparator_regs_set'
st_fprint_regs_set	Update the fprint ready and remaining registers. If an *overflow* has occurred, go to 'st_fifo_oflow_write', else go to 'st_oflow_count_inc_ack'

st_comparator_regs_set	Update the fprint ready and remaining registers. If an *underflow* has occurred, go to 'st_fifo_uflow_write', else go to 'st_oflow_count_dec_ack'
st_fifo_oflow_write	Write the overflow information to the output FIFO, go to 'st_oflow_status_set'
st_fifo_uflow_write	Write the underflow information to the output FIFO, go to 'st_oflow_status_reset'
st_oflow_status_set	Set the appropriate overflow status register, go to 'st_oflow_count_inc_ack'
st_oflow_status_reset	Reset all overflow status registers for the task, go to 'st_oflow_count_dec_ack'
st_oflow_count_inc_ack	Send an acknowledge signal to **fprint_registers** and return to 'idle'
st_oflow_count_dec_ack	Send an acknowledge signal to **comparator** and return to 'idle'
st_reset_task	Reset all counts and internal registers, go to 'st_oflow_reset_task_ack'
st_oflow_reset_task_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 now receives its fprint ready and fprint remaining signals from **oflow_registers**. It still 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.

The FSM that implements this submodule is described in Table 7. 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 7 - comparator FSM description

State	Description
idle	If fprints are ready or task has checked in go to 'st_set_task', otherwise stay in 'init'
st_set_task	Latch the Task ID of the ready/checked-in task, go to 'l st_load_pointer'
st_load_pointer	One clock cycle to fetch the tail pointer from **comp_registers**, go to 'st_load_fprint'
st_load_fprint	One clock cycle to fetch the fingerprints from **fprint_registers**, go to 'st_check_task_status'
st_check_task_status	If fingerprints are ready, go to 'compare_fprints', else if task has checked in then if fingerprints are remaining go to 'st_comparator_mismatch_detected' else go to 'st_fprint_reset_task'
st_compare_fprints	If the fingerprints match then go to 'st_comparator_inc_tail_pointer', otherwise go to 'st_comparator_mismatch detected'.
st_comparator_inc_tail_pointer	Send the inc tail pointer signal to **comp_registers** and wait for the acknowledge signal. Then go to 'st_comparator_count_dec'
st_comparator_count_dec	Send the count dec signal to **oflow_registers** and wait for the acknowledge signal. Then go to 'st_check_task_status'
st_comparator_mismatch_detected	Assert and latch the mismatch detected signal (will be reset when the state goes to 'idle') and go to 'st_fprint_reset_task'
st_fprint_reset_task	Send the reset signal to **fprint_registers** and wait for the acknowledge signal. Then go to 'st_comp_reset_task'
st_comp_reset_task	Send the reset signal to **comp_registers** and wait for the acknowledge signal. Then if a mismatch has been detected go to 'st_oflow_reset_task' else go to 'st_comparator_status_write'
st_oflow_reset_task	Send the reset signal to **oflow_registers** and wait for the acknowledge signal. Then go to 'st_comparator_status_write'
st_comparator_status_write	Assert the write status signals to **csr_registers**, and wait for the acknowledge signal. 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.
- Too much bus interference, hence switching to new design where each task stores only one fingerprint (infinite block size)
- This implies no more directory pointers and hence no more **comp_resigters**
- Send checkpoint interrupts to each core on a task when they have all completed the task
- Include a task success count register, so that the monitor will be informed only when a task has completed a programmed number of times, and thus implement software checkpointing.