Comparator with overflow 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**

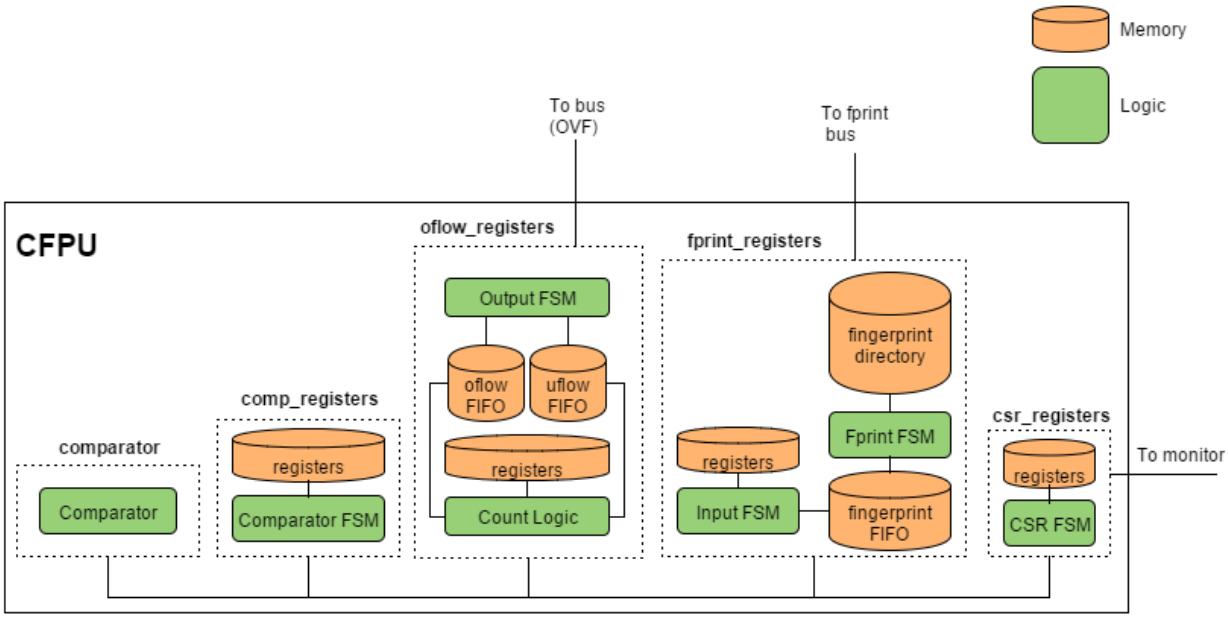

Figure 1 - Block level diagram of the CFPU

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	0x5000(L)D8	writedata(7:4) = Task ID writedata(3:0) = Core ID	This is a 16x2 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	0x500004(T) + 0x00000(L*4)0	writedata(9:0) = Pointer data	This is a 16 slot SRAM, indexed by Task ID, and the memory content is the directory start pointer for the task.
Directory End Pointer	Write	0x500008(T) + 0x00000(L*4)0	writedata(9:0) = Pointer data	This is a 16 slot SRAM, indexed by Task ID, and the memory content is the directory start pointer for the task.
Max Count Register	Write	0x50000CC	writedata(13:10) = Task ID writedata(9:0) = Max Count value	This is a 16 slot SRAM, indexed by Task ID, and the memory content is the maximum fingerprint count for each 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	-	16 bit register. If a task fails, the corresponding bit is set high

* L = Logical core ID (0 or 1)
* T = Task ID (0 to 15)

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	• *head_tail_data* – the start/end pointer data • *head_tail_offset* – the task id for which the pointer is being written • *set_head_tail* – a write signal for pointer data from **csr_registers** to **comp_registers** • *head_tail_ack* – an acknowledge of the write signal from **comp_registers** to **csr_registers** • *start_pointer_ex* – the start pointer corresponding to the task id in *'fprint_task_id'* • *end_pointer_ex* – the end pointer corresponding to the task id in *'fprint_task_id'* • *start_pointer_comp* – the start pointer corresponding to the task id in *'comp_task'* • *end_pointer_comp* – the end pointer corresponding to the task id in *'comp_task'*

fprint_registers	• *fprint_task_id* – the four bit task id which the incoming fingerprint (on the fprint bus) belongs to • *physical_core_id* – The four bit core id of the core that is sending the fingerprint • *logical_core_*id – the one bit logical core id from the Core Allocation Table corresponding to the above task id and core id
comparator	• *comp_status_write* – write signal from **comparator** to indicate task completion or failure • *comp_status ack* – pulse sent by **csr_registers** to acknowledge status write • *comp_task* – the four bit task id that the comparator is writing the status for • *comp_collision_dected* – this wire is 'high' is a mismatch in fingerprints has been detected
oflow_registers	• *csr_maxcount_write* – a write signal from **csr_registers** that indicates maxcount information is to be written • *csr_maxcount_writedata* – the maxcount value and task id for which the maxcount is being written (both fields are sent on this signal)

comp_registers

This submodule is in charge of keeping track of the head and tail pointers of the fingerprint directory for each task. It receives the start and end directory locations from **csr_registers**, and includes wrap around logic for both the head and tail pointers corresponding to these values.

There is a 16x2 bit 'fprint_ready' register that is indexed by Task ID and Logical ID, and the corresponding bit is asserted when there is a fingerprint ready (detected by condition head pointer leads the tail pointer).

There are three signals to **comparator** to indicate any discrepancy in the pointers:

- *head0_matches_head1* : head pointers of both logical cores are in sync
- *tail0_matches_head0* : logical core 0 has no fingerprints left.
- *tail1_matches_head1* : logical core 1 has no fingerprints left.

The functionality is handled by the FSM shown in Figure 2, and described in Table 4. Tail pointer increments are made independent of the FSM, from a signal sent directly from **comparator**.

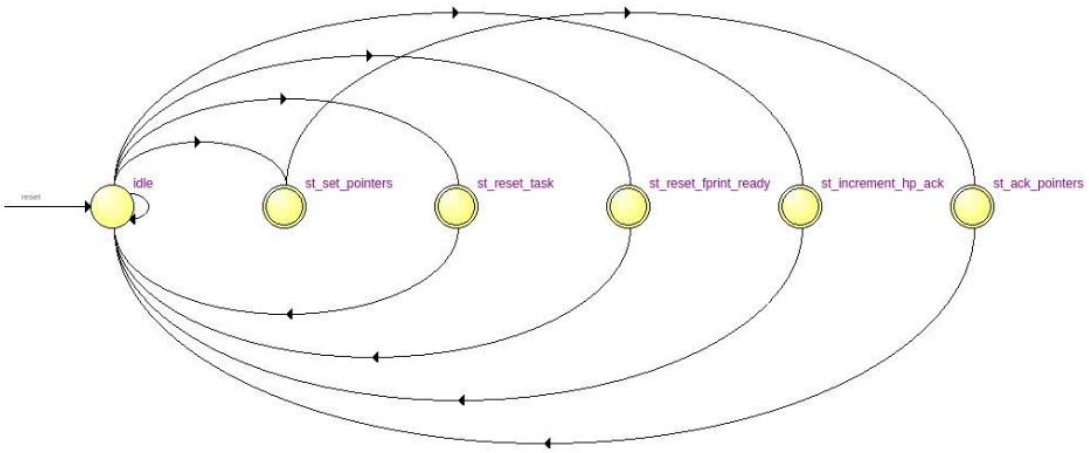

Figure 2 - comp_registers FSM

Table 4 - comp_registers FSM description

State	Description
idle	If **csr_registers** sends a write pointers signal, go to 'st_set_pointers. Else if **fprint_registers** sends an increase head pointer signal, go to 'st_increment_hp_ack'. Else if **comparator** sends a reset fprint ready signal, go to 'st_reset_fprint_ready'. Else if **comparator** sends a reset task signal, go to 'st_reset_task'. Otherwise stay in 'idle'
st_set_pointers	Set the head/tail pointer as provided by **csr_registers**, and go to 'st_ack_pointers'
st_ack_pointers	Send an acknowledge pulse to **csr_registers**, and go to 'idle'
st_increment_hp_ack	Increment the head pointer, send an acknowledge pulse to **fprint_registers**, and go to 'idle'
st_reset_fprint_ready	Reset the fprint_ready register bits corresponding to the **comparator** Task ID, and go to 'idle'
st_reset_task	Reset the fprint_ready register bits and the head and tail pointers to initial values corresponding to the **comparator** Task ID, 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 16x2 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.

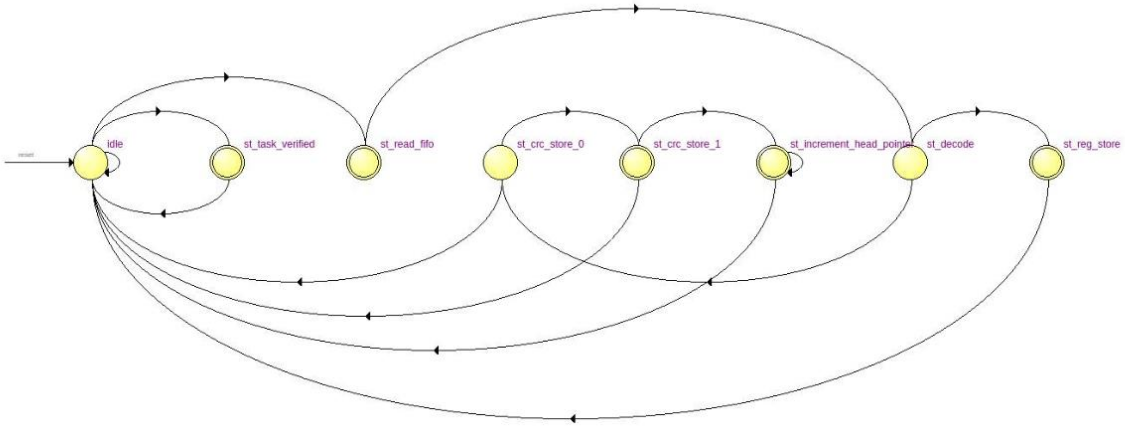

Figure 3 - fprint_registers FSM

Table 5 - fprint_registers FSM description

State	Description
idle	If the comparator sends a signal indicating a task has completed, go to 'st_task_verified'. Otherwise if the **fprint** bus FIFO is not empty, go to 'st_read_fifo'. Otherwise stay in 'idle'
st_read_fifo	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_reg_store', otherwise go to 'st_crc_store_0'
st_reg_store	Set the checkout/checkin register, go to 'idle'
st_crc_store_0	One clock cycle delay to wait for the head pointer of the task directory. Also check if the task has been checked out. If yes, go to 'st_crc_store_1', otherwise disregard the fingerprint and go to 'idle'
st_crc_store_1	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_increment_head_pointer'
st_increment_head_pointer	Send a signal to **comp_registers** to increase the directory head pointer. Wait for an acknowledge signal, and then go to 'idle'
st_task_verified	The comparator has sent a signal that the task is complete. Reset all the checkin and checkout register bits for both cores corresponding to that Task ID, send an acknowledge signal and return 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**.

The fingerprint count is maintained in an internal 16x2 slot SRAM, with 10 bits at each memory space to store the count value for the Task ID and corresponding Logical ID. The count is increased or decreased using the increase signal for the head and tail pointers respectively. Additional logic is incorporated to ensure that the count is not changed if the head and tail pointer are increased simultaneously for a single logical core.

This submodule contains a 16 bit overflow status register 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, and the overflowing core ID is stored in a physical ID table so that this ID is known at a later time when the fingerprint count decreases. An *underflow* occurs when the fingerprint count for both cores of an overflowing task (overflow status reg bit = 1) goes down to 0.

There are two FIFOs (oflow and uflow FIFOs) to track *overflow* and *underflow* events. When an *overflow* occurs, the Task ID and Core ID are written directly to the oflow FIFO. When an *underflow* occurs, the Task ID is written directly to the FIFO, along with the Core ID that was stored in the Physical ID table.

The contents of the oflow and uflow FIFOs are sent onto the **oflow** bus via the FSM shown in Figure 4, and described in Table 6. Priority is given to the contents of the oflow FIFO.

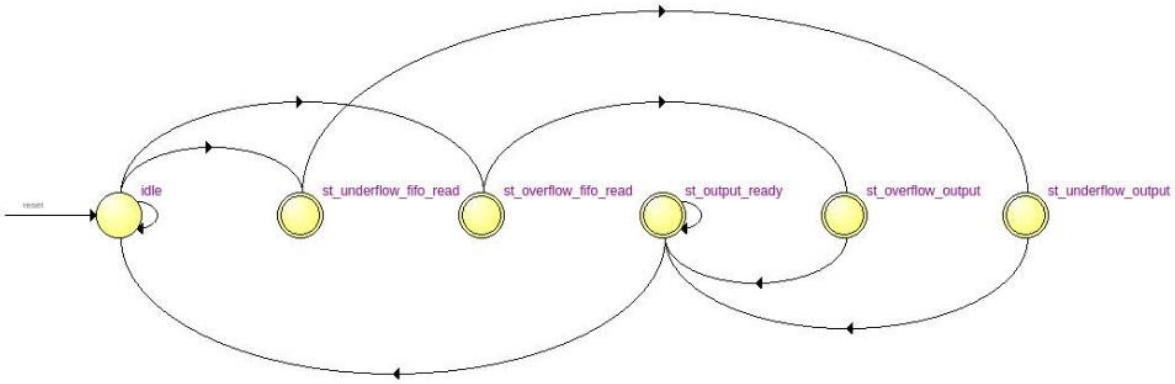

Figure 4 - oflow_registers output FSM

Table 6 - oflow_registers output FSM description

State	Description
idle	If the oflow FIFO is not empty, go to 'st_oflow_fifio_read', else if the uflow FIFO is not empty, go to 'st_uflow_fifio_read', else stay in idle
st_oflow_fifo_read	One clock cycle to fetch FIFO output, go to 'st_oflow_output'
st_uflow_fifo_read	One clock cycle to fetch FIFO output, go to 'st_uflow_output'
st_oflow_output	One cycle to latch the FIFO output on the outgoing writedata line, and go to 'output_ready'
st_uflow_output	One cycle to latch the FIFO output on the outgoing writedata line, and go to 'output_ready'
st_output_ready	Wait for the incoming **oflow** bus waitrequest signal to go low, and then go to 'idle'

comparator

This submodule is responsible for comparing fingerprints and writing the task completion/failure status to **csr_registers**. It receives an fprint ready signal for all tasks from **comp_registers** indicating when two fingerprints are ready to be compared. 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.

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

Table 7 - comparator FSM description

State	Description
init	If fprints are ready or task has checked in go to ‘set_task’, otherwise stay in ‘init’
set_task	Latch the Task ID of the ready/checked-in task, go to ‘load_pointer’
load_pointer	One clock cycle to fetch the tail pointer from **comp_registers**, go to ‘load_fprint’
load_fprint	One clock cycle to fetch the fingerprints from **fprint_registers**, go to ‘check_task_status’
check_task_status	If fingerprints are ready, go to ‘compare_fprints’, else if task has checked in then go to ‘task_complete’
task_complete	If the head pointers of both cores do not match, go to ‘mismatch_detected’, otherwise go to ‘reset_fprint_ready’
compare_fprints	If the fingerprints match then go to ‘increment_tail_pointer’, otherwise go to ‘mismatch detected’
mismatch_detected	Assert and latch the comp_collision_detected signal (will be reset when the state goes to ‘init’) and go to ‘task_verified’
task_verified	Send the task verified signal to **fprint_registers** and wait for the acknowledge signal. Then, if a mismatch is detected go to ‘st_reset_task’, otherwise go to ‘write_status_reg’
increment_tail_pointer	Send the increment tail pointer pulse to **comp_registers**, and go to ‘check_if_done’
check_if_done	If there are fingerprints remaining then go to ‘compare_fprints’ else go to ‘reset_fprint_ready’
reset_fprint_ready	Send the reset fprint ready signal to **comp_registers**, and wait for the acknowledge signal. Then, if the task has checked in or a mismatch is detected go to ‘task_verified’, else go to ‘init’
st_reset_task	Send the reset task signal to **comp_registers** and wait for the acknowledge signal. Then go to ‘write_status_reg’
write_status_reg	Assert the write status signals to **csr_registers**, and wait for the acknowledge signal. Then, go to ‘init’

Changes to be made

- Include TMR support
- Simplify register programming on the **csr** bus (get rid of data in the address field)
- Separate directory space for each logical core on the same task. Right now they are set to the same start and end pointer as specified by the task. Change this so that each logical core can use a different area of its directory
- As a result of separate directory, the pointer matching signals from **comp_registers** to **comparator** will be invalid. Therefore, change the logic so that remaining fingerprint discrepancies are handled by the fingerprint count from **oflow_registers**