LegUp fifo_read implementation

Related signals

- RVD Interface
  - `ready_out, valid_in, data_in`
- “Pre-fetch” registers
  - `consumed_data, c_valid, c_taken`
- Enable condition
  - `enable_cond`
- Stall signal
  - `stall_signal`

FIFO Read:

```
// RVD Interface.
assign ready_out = ~c_valid | c_taken;
always @ (posedge clk) begin
  if (ready_out & valid_in) consumed_data <= data_in;
  if (ready_out & valid_in) consumed_valid <= 1;
  else if (c_taken) consumed_vaid <= 0;
End

// Stall and flow.
always @ (*) begin
  stall_signal = 0;
  if (enable_cond & ~c_valid) stall_signal = 1;
  if (…other conditions causing stall…) stall_signal = 1;
end
assign c_taken = enable_cond & ~stall_signal;
```

LegUp fifo_write implementation

Related signals

- RVD Interface
  - `ready_in, valid_out, data_out`
- Enable condition
  - `enable_cond`
- Stall signal
  - `stall_signal`
- Two helper signals…
  - `data_not_taken_when_stall_last_cycle`
  - `stalln_reg`

FIFO Write:

```
// The helper registers.
always @ (posedge clk) begin
  data_not_taken_when_stall_last_cycle <=
                              stall_signal & valid_out & ~ready_in;
  stalln_reg <= ~stall_signal;
end

// Stall and flow.
always @ (*) begin
  stall_signal = 0;
  if (enable_cond & valid_out & ~ready_in) stall_signal = 1;
  if (…other conditions causing stall…) stall_signal = 1;
end
assign data_out = data_path_register;
assign valid_out = enable_cond &
    (data_not_taken_when_stall_last_cycle | stalln_reg);
```

Scheduling of fifo_write

- The data port has to be directly connected to registers
  - Prevents the case that the data is not valid (due to other stalls) at the state where fifo_write is scheduled
  - e.g., `fifo_write(output_fifo, fifo_read(input_fifo) + x);`
  - If fifo_read and fifo_write are scheduled in the same cycle, the above example becomes messy to handle
- Changed scheduler to avoid chaining of fifo_write data argument