Pattern Generator (with SCARF interface)

The Pattern Generator allows for a selectable number of IO pins to drive out high/low values which correspond to one/zero values in an external memory. The external memory is read and the values are then driven to IO pins. The goal is to define the shape of the pattern in the byte data contained in the external memory.

There are two configuration values used to define the rate at which the IO pin values are changed. The first one is called stage1_count, which is a counter value to help get your FPGA board's unique clock frequency close to a standard real time period. My FPGA board clock is 12MHz, so I write a decimal 12 to the stage1_count. A count of 12 12MHz periods is 1us. If you had a 8MHz clock, you would write a value of 8 which would get you to 1us. The second configuration value is called timebase. The timebase selects powers of 10 to increase the stage1 to a much slower period. A timebase of zero will keep us at 1us, but a time base of 1 would move us to 10us. A timebase of 6 would increase the stage1 period to 1 second (10 ^ 6 * stage1_count * fpga_clk_period).

As for the number of IO pins, a 2bit configuration value allows for 1, 2, 4 or 8 pins to be selected. If 1 pin is selected, the external memory will be read once every 8 times the single pin is updated (8bits in a byte). If 2 pins are selected the external memory will be read every 4th time the pins are updated. If 8 pins are selected, the external memory needs to be read each time the pins are updated (each bit the in byte corresponds to 1 of the 8 IO pins). If you wanted to drive 5 or 7 pins, you would need to select 8 pins and not connect the unused pins.

A 1 pin configuration will index the byte bits starting with the MSB down to the LSB. A 2 pin configuration will take the 2 MSB at a time (not separate nibbles). A 4 pin configuration will start with the upper nibble and end with the lower nibble.

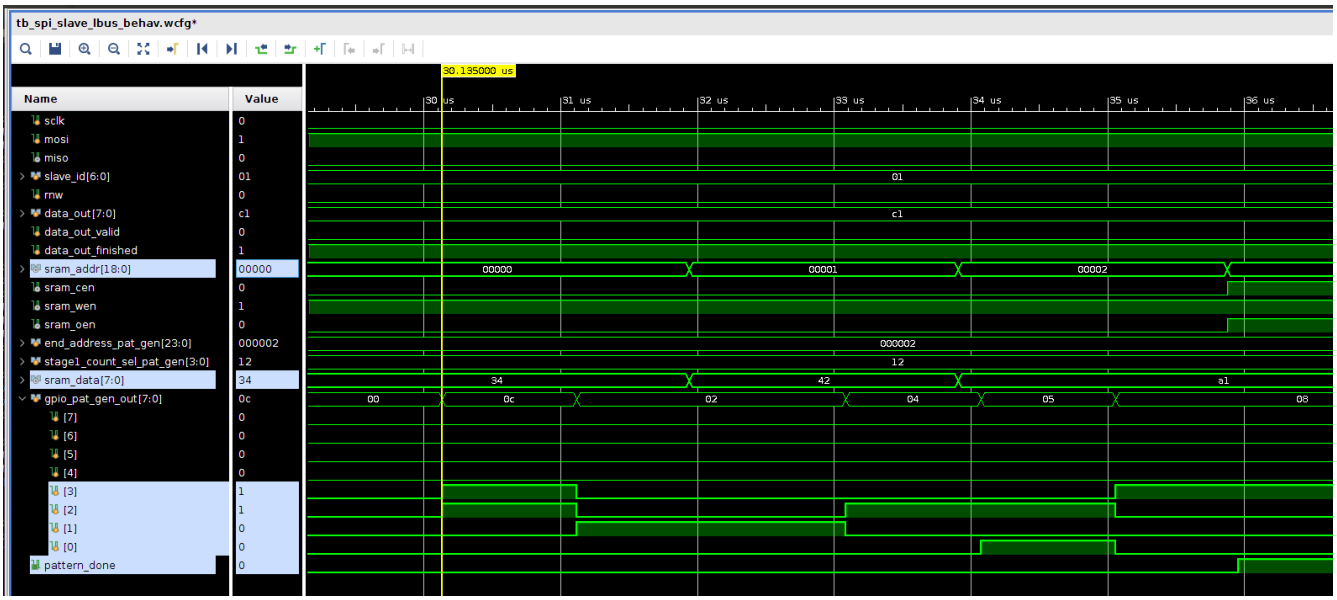
Many times software bit-banging is an easier way to drive a pattern. However if precision is needed, the FPGA pattern generator is a great tool. An example of this is driving JTAG stuck-at-fault scan patterns versus at-speed scan patterns. Stuck-at-fault scan patterns can be reliably bit-banged, but at-speed patterns require moments of precise timing.

```
module pattern_gen (
    input  logic        clk,
    input  logic        rst_n,
    input  logic        enable_pat_gen,
    input  logic [23:0] end_address_pat_gen, // address 24'd0 is always the start address
    input  logic  [1:0] num_gpio_sel_pat_gen, // 2'b00 is 1, 2'b01 is 2, 2'b10 is 4 and 2'b11 is 8
    input  logic  [2:0] timestep_sel_pat_gen, // 10 ^ n
    input  logic  [3:0] stage1_count_sel_pat_gen, // 4'd12 for 12MHz clock, 4'd8 for 8MHz
    input  logic        repeat_enable_pat_gen,
    output logic        pattern_active,
    output logic        pattern_done,
    output logic  [7:0] gpio_pat_gen_out,
    input  logic  [7:0] sram_data,
    output logic [18:0] sram_addr_pat_gen
    );
```

Red signals are configuration from SCARF regmap. green signals are FPGA IO, orange signals are for external SRAM and will drive glue logic which shares the external SRAM with other designs.

Below is a waveform diagram that shows how sram values are driven onto pins. I this case 4 output pins are selected. The FPGA clock is 12MHz, which means a value of decimal 12 is written to

stage1_count_sel[3:0]. The stop address is 0x000002, so only 3 bytes will be driven. Pattern repeat is disabled so the pattern_done signal goes high after the 3 byte has been driven. Each 1us a nibble of sram data driven to the pads (2 * 3 = 6 nibbles in 6us).



This simulation is included as tb_scarf.sv. I ran it in the vivado behavioral simulator (free version).