SCARF - “Scarf Connects A Raspberry pi to Fpga” **June 6, 2020**

SCARF is a bare-bones SPI slave written in systemverilog, which clock-crosses input and output data to the FPGA clock domain. It’s goal is to provide a simple, uniform interface to FPGA designs, allowing them to be easily controlled from a single external SPI master (A Raspberry pi).

SCARF comes with python drivers and example code.

Whether you have a single design to implement in the FPGA or several, they all can share a single 4pin SPI interface. For simplicity the SPI slave works only with CPOL=0 and CPHA=0.

Why use SCARF? If you want to send data to/from your FPGA using a Raspberry Pi, the SPI bus is fast and has low overhead. The SCARF block is less than 50 flipflops and the interface is very simple. Once you see how easy it is to interface to your FPGA designs, say good-bye to custom interfaces. SCARF has a mechanism for individual designs to respond to a generic query command, which can be used to see all SCARF slaves on the bus. In my bench FPGA I have 20 or more designs implemented and SCARF keeps them all organized for me, just as if they were all individual ICs connected on a circuit board.

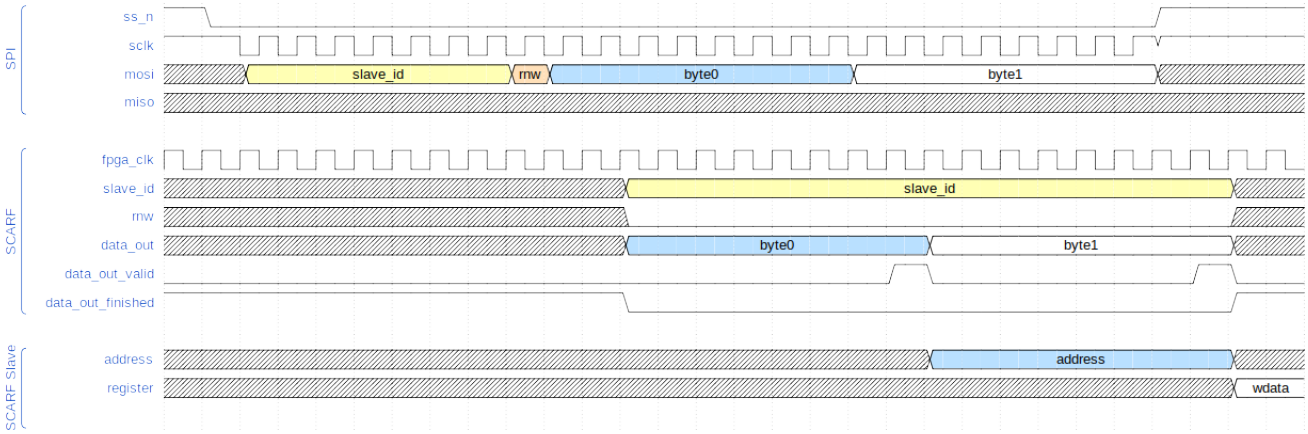
Here is what the SCARF instance looks like:

```
scarf u_scarf
    ( .sclk,               // input
      .mosi,               // input
      .miso,               // output
      .ss_n,               // input
      .clk,                // input
      .rst_n,              // input
      .read_data_in,       // input  [7:0]
      .rst_n_sync,         // output
      .data_out,           // output [7:0]
      .data_out_valid,     // output
      .data_out_finished,  // output
      .slave_id,           // output [6:0]
      .rnw                 // output
     );
```

The RED signals are the SPI bus (these are FPGA pins), the GREEN signals are FPGA clock and reset (these are FPGA pins), and the BLUE signals are the SCARF interface (these are not FPGA pins, they are internal signal). The BLUE signals interface to your unique FPGA designs.

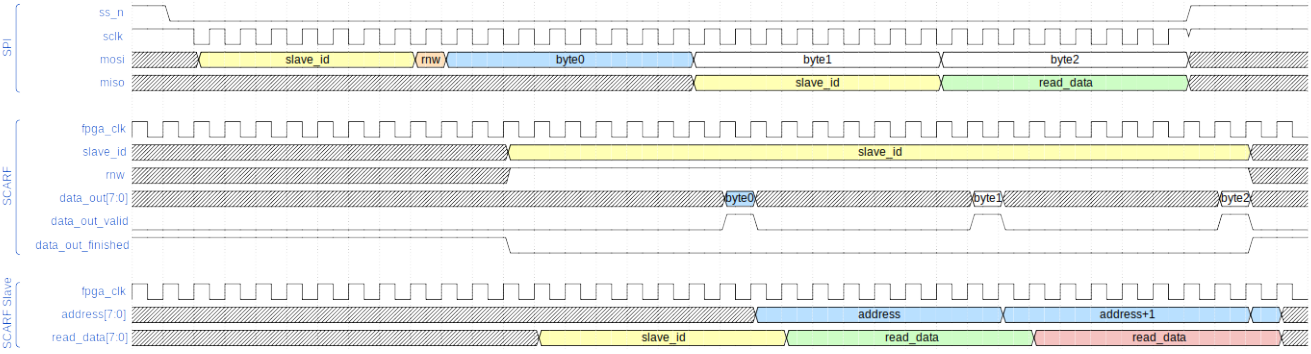
The SCARF protocol (FPGA side) uses the first data byte in the SPI data stream to send a slave address and read-not-write bit. If you have multiple designs in your FPGA the slave address will allow each design to know if the data is directed to it or not. The read-not-write bit will indicate the direction of data flow.

In the example below a single byte address has a single byte of data written to it.



The first SPI data byte contains the 7bit “slave_id” followed by a “rnw” (read-not-write) bit. After that anything goes. The SCARF block will drive the slave_id and rnw bit until it detects that the SPI bus cycle has completed. A single byte width “data_out” output is valid to be captured when the “data_out_valid” pulses high for a single fpga_clk period. The “data_out_finished” signal will indicate that there is no more data expected.

In the example below a single byte address has a single byte of data read from it. Due to the clock crossing the 4th SPI byte will carry the first read data byte. The 3rd byte will carry the echoed slave_id as an acknowledgment that the design is present and responding.



One of the included sample designs is a pattern generator that reads from an external SRAM and drives FPGA output pins. Two separate slave_ids are used, one for the external SRAM interface and the other for the pattern generator register map.

The slave_id that controls the external sram will use 3 bytes for address and a single byte for data. Address auto-increment is used such that a single SPI bus cycle can write or read many bytes of sram data. The slave_id which controls the pattern generator needs only a single byte of address as the register map only has a handful of addresses.

This pattern generator design has a SCARF interface and uses a unique slave_id:

```
scarf_pattern_generator
   # ( .SLAVE_ID(7'h01) )
   u_scarf_pattern_generator
```

```
    ( .clk,                                     // input
      .rst_n_sync,                              // input
      .data_in          (data_out),             // input [7:0]
      .data_in_valid    (data_out_valid),       // input
      .data_in_finished (data_out_finished),    // input
      .slave_id,                                // input [6:0]
      .rnw,                                     // input
      .read_data_out    (read_data_out_pat_gen), // output [7:0]
      .pattern_active,                          // output
      .pattern_done,                            // output
      .gpio_pat_gen_out,                        // output [7:0]
      .sram_data,                               // input  [7:0]
      .sram_addr_pat_gen                        // output [18:0]
      );
```

The signals in BLUE are the SCARF interface. All other signals are needed for the pattern generator design. The specific slave_id used is being passed to the instance as a parameter override.

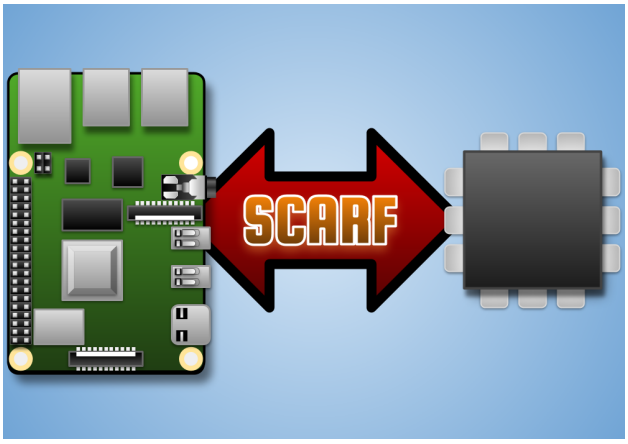
Here is an interface block that converts SCARF to external SRAM bus cycles:

```
scarf_ext_sram
   # ( .SLAVE_ID(7'h02) )
   u_scarf_ext_sram
     ( .clk,                                    // input
       .rst_n_sync,                             // input
       .data_in             (data_out),         // input [7:0]
       .data_in_valid       (data_out_valid),   // input
       .data_in_finished    (data_out_finished), // input
       .slave_id,                               // input
       .rnw,                                    // input
       .read_data_out       (read_data_out_sram), // output [7:0]
       .sram_data_in        (sram_data),        // input  [7:0]
       .sram_data_out       (sram_wdata),       // output [7:0]
       .sram_addr           (sram_addr_scarf),  // output [18:0]
       .sram_oen            (sram_oen_scarf),   // output
       .sram_wen            (sram_wen_scarf),   // output
       .sram_cen            (sram_cen_scarf)    // output
      );
```

As SCARF data is written, or data is read from this design, the interface to external SRAM will be exercised. What is not shown here is that the external SRAM pins are being controlled from two separate designs… scarf_ext_sram and scarf_pattern_generator. There is glue logic to allow both designs to access the common SRAM resource at different times. The scarf_ext_sram design is used to load the external SRAM with pattern data (from the Raspberry Pi).

By using the SCARF interface, FPGA IO pins that were previously used to trigger/enable design actions or indicate design status may be transferred to memory mapped IO (register map). If real-time status needs to be communicated, a common interrupt pin may be implemented.

In summary, SCARF is a simple way to consolidate FPGA IO into a single SPI interface. Just as USB killed the parallel port, SCARF is going to kill custom interfaces and allow you to have many designs coexist inside the FPGA.


SCARF