AN 923: Routing Intel® Stratix® 10 HPS Peripherals to FPGA Fabric

Updated for Intel® Quartus® Prime Design Suite: 20.3
## Contents

1. AN 923: Routing Intel® Stratix® 10 HPS Peripherals to FPGA Fabric................................. 3
   1.1. Intel Stratix 10 HPS Peripherals that Support Routing to the FPGA........................ 3
   1.2. Design Example: Intel Stratix 10 HPS IP Interface to FPGA....................................... 5
      1.2.1. Prerequisites................................................................. 5
      1.2.2. Getting Started............................................................ 6
      1.2.3. Generating the Initial HDL in Platform Designer................................................ 6
      1.2.4. Modifying Top Level File................................................. 9
      1.2.5. Adding Pin Assignments for SPIM0.................................... 9
      1.2.6. Hardware Programming File Compilation and Generation................................. 9
      1.2.7. Building U-Boot......................................................... 9
      1.2.8. Preparing QSPI Image.................................................... 10
      1.2.9. Building Linux......................................................... 10
      1.2.10. Building Yocto Rootfs.................................................. 11
      1.2.11. Building spidev Test Program........................................ 11
      1.2.12. Creating SD Card Image................................................ 12
      1.2.13. Booting the Board..................................................... 12
      1.2.14. Testing the SPIM0..................................................... 13
   1.3. Document Revision History for the AN 923: Routing Intel Stratix 10 HPS Peripherals to FPGA Fabric................................................................. 13
1. AN 923: Routing Intel® Stratix® 10 HPS Peripherals to FPGA Fabric

The Intel® Stratix® 10 SoC device families integrate an Arm® Cortex-A53-based hard processor system (HPS) consisting of processor, peripherals, and memory interface with the FPGA fabric using a high-bandwidth interconnect backbone. The Intel Stratix 10 HPS interface provides up to 48 I/O pins to share with multiple peripherals through sets of configurable multiplexers. This application note describes the steps required to route an HPS peripheral through the FPGA interface using Platform Designer and Intel Quartus® Prime Pro Edition software version software. A simple design example is included to demonstrate exporting HPS SPI/M0 peripheral signals to the FPGA interface using an Intel Stratix 10 SoC Development Kit.

1.1. Intel Stratix 10 HPS Peripherals that Support Routing to the FPGA

The following types of Intel Stratix 10 HPS peripherals are capable of routing to the FPGA fabric:

- Secure Digital/Multimedia Card (SD/MMC)
- Ethernet Media Access Controller (EMAC)
- Serial Peripheral Interface (SPI)
- Universal Asynchronous Receiver/Transmitter (UART)
- Inter-Integrated Circuit (I²C)
- NAND Flash Controller
- TRACE Interface

In many cases, routing the HPS IP signals to the FPGA external interface allows more signals to be exposed.

Table 1. Peripherals that Support Signal Routing from the HPS Domain to FPGA Domain

The following table lists the interface type that is available depending on whether the IP interface is pinned out in the HPS domain or the FPGA domain.

<table>
<thead>
<tr>
<th>Peripherals</th>
<th>HPS Domain</th>
<th>Interface Description</th>
<th>FPGA Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD/MMC</td>
<td></td>
<td>Standard SD/MMC interface with up to 8-bit data bus</td>
<td>Standard SD/MMC interface, including:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Up to 8-bit data bus</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Card detect interface</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Card interrupt</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>Peripherals</th>
<th>Interface Description</th>
<th>HPS Domain</th>
<th>FPGA Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Voltage switching</td>
<td>MII, GMII, RMII, RGMII, SGMII</td>
</tr>
<tr>
<td>EMAC</td>
<td>RMII and RGMII Interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPI Master</td>
<td>MOSI/MISO SPI interface configurable to single or dual</td>
<td>MOSI/MISO SPI interface</td>
<td>Migration enabled that support up to four slaves; interface achieved by</td>
</tr>
<tr>
<td></td>
<td>slaves</td>
<td>interface with output</td>
<td>connecting the signals to bidirectional buffers</td>
</tr>
<tr>
<td>SPI Slave</td>
<td>MOSI/MISO SPI interface configurable to single master</td>
<td>MOSI/MISO SPI interface</td>
<td>Migration enabled to single master</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interface with output</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>output enables</td>
<td></td>
</tr>
<tr>
<td>UART</td>
<td>Standard UART interface with flow control signals</td>
<td>Standard UART interface</td>
<td>Migration enabled to support the signals flow control, including DSR, DCD,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interface with flow</td>
<td>RI, and DTR; and two user-defined output signals are also available</td>
</tr>
<tr>
<td></td>
<td></td>
<td>control signals</td>
<td></td>
</tr>
<tr>
<td>I²C</td>
<td>Standard I²C interface</td>
<td>Standard I²C interface</td>
<td>Migration enabled to support the signals to a bidirectional buffer</td>
</tr>
<tr>
<td>NAND Flash Controller</td>
<td>Standard NAND Interface with 8- or 16-bit of data and one set of CE and R/B signals</td>
<td>Standard NAND Interface with 8- or 16-bit of data and four sets of CE and R/B signals</td>
<td></td>
</tr>
<tr>
<td>TRACE</td>
<td>TRACE debug interface</td>
<td>TRACE debug interface</td>
<td></td>
</tr>
</tbody>
</table>

For a description of each of the peripheral signal interfaces listed above, refer to the corresponding chapters in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.

**Related Information**


---

(1) The SD/MMC controller does not directly support reset, voltage switching, card interrupts, power enable or write protect functions. However, you can connect these signals to general-purpose I/Os (GPIOs).
1.2. Design Example: Intel Stratix 10 HPS IP Interface to FPGA

This design example, based on the Golden System Reference Design (GSRD), uses the Intel Stratix 10 SoC Development Kit resources to demonstrate the routing of the Intel Stratix 10 HPS SPIM0 peripheral signals to the FPGA interface.

Figure 1. High-level Routing Layout of Intel Stratix 10 SoC Board Design Example

The following sections, in this document, provide the necessary information to route the HPS peripherals to the FPGA interface.

1.2.1. Prerequisites

This design example is based on the Intel Stratix 10 GSRD and tested with Intel Quartus Prime Pro Edition software version 20.3.

Before starting this design example, refer to the following web pages:

- Intel Stratix 10 SX SoC Development Kit
- Intel Stratix 10 SoC GSRD

Related Information

- Intel Stratix 10 SoC Development Kit
  For more information about the Intel Stratix 10 SoC Development Board, documentation, and installation files
- Intel Stratix 10 SoC GSRD
  For more information, refer to the following sections: Prerequisites, GHRD Overview, Running GSRD with Pre-Built Binaries, and Build Flow.

1.2.1.1. Hardware Requirements

The hardware required for this design example is:

- Intel Stratix 10 SoC Development Kit
- 4 GB DDR4 HILO memory card
- SD/MMC HPS Daughtercard
- SDM QSPI Bootcard (MT25QU02G)
• Additional SDM QSPI Bootcard - connected to FPGA to be accessed over SPI
• Mini USB cable for serial output
• Micro USB cable for on-board Intel FPGA Download Cable II

1.2.1.2. Software Requirements

The software required for this design example are:
• Intel Quartus Prime Pro Edition software version 20.3 and above
• Intel Stratix 10 GSRD downloaded from RocketBoards

Related Information
• Intel Stratix 10 SoC GSRD
• Download Center for FPGAs on the Intel website

1.2.2. Getting Started

1. Create a top folder for this example, as the rest of the commands use this location.

```
mkdir ~/s10_spim_fpga
```

Compilation of the software relies on downloading the build toolchain and adding it to the PATH variable.

```
cd ~/s10_spim_fpga
wget https://releases.linaro.org/components/toolchain/binaries/7.5-2019.12/aarch64-linux-gnu/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu.tar.xz
tar xf gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu.tar.xz
export PATH=`pwd`/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu/bin:$PATH
export ARCH=arm64
export CROSS_COMPILE=aarch64-linux-gnu-
```

2. Retrieve a copy of the hardware design.

```
cd ~/s10_spim_fpga
wget https://releases.rocketboards.org/release/2020.05/gsrd/s10_gsrd/s10_soc_devkit_ghrd.tar.gz
tar xf s10_soc_devkit_ghrd.tar.gz
```

1.2.3. Generating the Initial HDL in Platform Designer

1. In the Quartus Prime Pro Edition navigation bar, select Tools ➤ Platform Designer.
2. In the Platform Designer window, select File ➤ Open ➤ qsys_top.qsys.
3. In the System View tab, double click on s10_hps to open the HPS Parameters window.
4. Go to Pin Mux and Peripherals ➤ Advanced ➤ Advanced FPGA placement and select the Route to FPGA box for SPIM0 to Yes, then click on Apply Selections.

5. Double click in the Export column to export the s10_hps.spim0 conduit.

6. Double click in the Export column to export the s10_hps.spim0_sclk_out conduit.

7. Add a new GPIO soft IP to use as SPI chip select.

8. Add a new component of type PIO (Parallel I/O) Intel FPGA IP, name it spi_cs_pio and configure it as follows:
   - **Width**: 1
   - **Direction**: bidir
   - **Enable individual bit setting/clearing**: yes
   - **Output Port Reset Value**: 0x0000000000000001
9. Change the base address of `spi_cs_pio.s1` to 0x0001_0000 to avoid conflicts with other IPs.

10. Connect the `spi_cs_pio` as follows:
   - `clk`: to `clk_100.out_clk`
   - `reset`: to `rst_in.out_reset`
   - `s1`: to `s10_hps.h2f_lw_axi_master`

   Double click on the `spi_cs_pio.external` connection to export it and create the conduit `spi_cs_pio.spi_cs_pio_external_connection_export`.

11. Go to **Generate ➤ Show Instantiation Template**, and save the template in a file, so you have the new signal names for `qsys_top`.

12. Click on the **Generate** button to generate the system.
1.2.4. Modifying Top Level File

1. In the `ghrd_s10_top.v`, add the top level signals connecting to SPI.

   *Note: Only a partial view displayed, below.*

   ```vhd
   module ghrd_s10_top (
   output wire spi_clk,
   output wire spi_cs,
   output wire spi_mosi,
   input  wire spi_miso,
   )
   ```

2. Connect the relevant Platform Designer exported signals to the top level signals:

   *Note: Only a partial view displayed, below.*

   ```vhd
   module ghsys_top soc_inst (  
   .s10_hps_spim0_sclk_out_clk            (spi_clk),  
   .s10_hps_spim0_spi_cs_pio_external_connection_export (spi_cs),  
   .s10_hps_spim0_spi_mosi_o                  (spi_mosi),  
   .s10_hps_spim0_spi_miso_i                  (spi_miso),  
   .s10_hps_spim0_spi_ss_in_n                 (1'b1),
   )
   ```

1.2.5. Adding Pin Assignments for SPIM0

In ghrd_1sx280lu2f50e2vg.qsf, connect the top level signals to the correct pins on the Intel Stratix 10 SoC development kit, and set them to 1.8V:

```vhd
set_location_assignment PIN_BD28 -to spi_clk
set_location_assignment PIN_BF26 -to spi_cs
set_location_assignment PIN_BC28 -to spi_mosi
set_location_assignment PIN_BE27 -to spi_miso
set_instance_assignment -name IO_STANDARD "1.8 V" -to spi_clk -entity
ghrd_s10_top
set_instance_assignment -name IO_STANDARD "1.8 V" -to spi_cs -entity
ghrd_s10_top
set_instance_assignment -name IO_STANDARD "1.8 V" -to spi_mosi -entity
ghrd_s10_top
set_instance_assignment -name IO_STANDARD "1.8 V" -to spi_miso -entity
ghrd_s10_top
```

1.2.6. Hardware Programming File Compilation and Generation

After the Platform Designer system is set up, the top level RTL file is updated, the related signal pin location is assigned and timing constrained, the design can be compiled and the SOF programming file generated.

In the Quartus Prime Pro Edition navigation bar, select Processing ➤ Start Compilation to generate the SOF programming file.

1.2.7. Building U-Boot

```bash
cd ~/s10_spim_fpga
rm -rf u-boot-socfpga
git clone https://github.com/altera-opensource/u-boot-socfpga
cd u-boot-socfpga
git checkout -b test-bootloader -t origin/socfpga_v2020.04
make clean && make mrproper
make socfpga_stratix10_defconfig
make -j 24
```
1.2.8. Preparing QSPI Image

The following commands must be run inside an Intel Quartus Prime shell. Start the Intel Quartus Prime shell with the following command:

```
~/intelFPGA_pro/20.3/embedded/embedded_command_shell.sh
```

```
cd ~/s10_spim_fpga
quartus_pfg -o s10_soc_devkit_grd/output_files/grd_1sx280lu2f50e2vg.sof flash_image.jic
-o device=MT25QU128
-o flash_loader=1SX280LU3
-o hps_path=u-boot-socfpga/spl/u-boot-spl-dtb.hex
-o mode=ASX4
```

1.2.9. Building Linux

1. Clone the Linux git tree and get the LTSI branch:

```
cd ~/s10_spim_fpga/
git clone https://github.com/altera-opensource/linux-socfpga
  linux-socfpga.a53
cd linux-socfpga.a53
git checkout -b test-kernel -t origin/socfpga-5.4.44-lts
make clean && make mrproper
```

2. Change the kernel configuration by editing `arch/arm64/configs/defconfig` as follows:

```
CONFIG_SPI_SPIDEV=y
CONFIG_GPIO_ALTERA=y
CONFIG_SPI_DESIGNWARE=y
CONFIG_SPI_DW_MMIO=y
```

3. Edit the device tree file: `arch/arm64/boot/dts/altera/socfpga_stratix10_socdk.dts` as follows:

Add the FPGA clock that is used by the FPGA fabric:

```
clk_100: clk_100 {
    compatible = "fixed-clock";
    clock-frequency = <100000000>;
    clock-output-names = "clk_100-out_clk";
};
```

Add the PIO module which controls the SPI CS signal:

```
spi_cs_pio: gpio@0xf9010000 {
    compatible = "altr,pio-19.1", "altr,pio-1.0";
    reg = <0xf9010000 0x00000020>;
    clocks = <&clk_100>;
    altr,gpio-bank-width = <1>;
    #gpio-cells = <2>;
    gpio-controller;
    status = "okay";
};
```

Enable SPIM0, make it use the soft IP PIO as chip select, and enable it to be used by the spidev driver:

```
&spi0 {
    num-cs = <1>;
    cs-gpios = <&spi_cs_pio 0 0>;
    status = "okay";
    spidev@0 {
```
compatible = "custom,spidev";
reg = <0x0>;
spi-max-frequency = <0x5F5E10>;
enable-dma = <0x1>;
};
};

4. Edit the spidev driver source file: (drivers/spi/spidev.c) for the "compatible" property to match the created device tree entry.

{ .compatible = "lineartechnology,ltc2488" },
{ .compatible = "ge,achc" },
{ .compatible = "semtech,sx1301" },
{ .compatible = "custom,spidev" },
{}

5. Build Linux kernel and device trees.

make clean && make mrproper
# enable JFFS2 and disable 4K sectors for booting from QSPI
echo "CONFIG_JFFS2_FS=y" >> arch/arm64/configs/defconfig
echo "CONFIG_MTD_SPI_NOR_USE_4K_SECTORS=n" >> arch/arm64/configs/defconfig
# reduce QSPI clock to work on early boards
sed -i 's/spi-max-frequency = <100000000>;/
spi-max-frequency = <50000000>;/g'
arch/arm64/boot/dts/altera/socfpga_stratix10_socdk.dts
make defconfig
make -j 48 Image dtbs modules
make modules_install INSTALL_MOD_PATH=modules_install
rm -rf modules_install/lib/modules/*/build
rm -rf modules_install/lib/modules/*/source
cd ..

1.2.10. Building Yocto Rootfs

cd $ROOTFS_TOP
rm -rf s10 && mkdir s10 && cd s10
git clone -b zeus git://git.yoctoproject.org/poky.git
git clone -b master git://github.com/kraj/meta-altera.git
source poky/oe-init-build-env ./build
echo 'MACHINE = "stratix10"' >> conf/local.conf
echo 'IMAGE_FSTYPES = "tar.gz"' >> conf/local.conf
echo 'BBLAYERS += "$TOPDIR/../../../meta-altera */"' >> conf/bblayers.conf
bitbake core-image-minimal
ln -s $ROOTFS_TOP/s10/build/tmp/deploy/images/stratix10/
core-image-minimal-stratix10.tar.gz $LINUX_BIN/a53/

1.2.11. Building spidev Test Program

The test application is very simple, it creates a single SPI transfer composed of two buffers:

- One 1-byte TX buffer containing the flash Read ID command
- One 6-byte RX buffer containing the response from the flash to the Read ID command
1. Go to the Accessing SPI Devices in Linux web page and scroll to the bottom of the page to the demo application that shows how to read the Flash ID from a SPI Flash device.

2. Copy-paste the contents and save it as: `spidev_flash_test.c`.

3. Compile the above C source code file:
   
   ```bash
cd ~/s10_spim_fpga
$(CROSS_COMPILE)gcc -static spidev_flash_test.c -o spidev_flash_test
```

1.2.12. Creating SD Card Image

   ```bash
cd STOP_FOLDER/
sudo rm -rf sd_card && mkdir sd_card && cd sd_card
wget https://releases.rocketboards.org/release/2020.05/gsrd/tools/make_sdimage_p3.py
chmod +x make_sdimage_p3.py
mkdir fatfs && cd fatfs
cp $LINUX_BIN/a53/Image .
cp $LINUX_BIN/a53/socfpga_stratix10_socdk.dtb .
cp ../../u-boot-socfpga/u-boot.img .
cd ..
mkdir rootfs && cd rootfs
sudo tar xf $LINUX_BIN/a53/core-image-minimal-stratix10.tar.gz
sudo rm -rf lib/modules/*
sudo cp ../../u-boot-socfpga/u-boot.img .
cd ..
sudo python3 make_sdimage_p3.py -f
   -P fatfs/*,num=1,format=fat32,size=100M
   -P rootfs/*,num=2,format=ext3,size=400M
   -s 512M
   -n sdcard_s10.img
```

1.2.13. Booting the Board

1. Write the SD card image to the SD card and insert the SD card in the slot.

2. Configure the Intel Stratix 10 SoC Development Kit as follows:
   
   - SW1: 1: OFF, rest: ON
   - SW2: 1: ON 2: ON 3: ON 4: OFF (set MSEL to JTAG)
   - SW3: all OFF
   - SW4: 1: ON 2: OFF 3: OFF 4: ON

3. Use Intel Quartus Prime Programmer to program the QSPI flash:
   
   ```bash
cd STOP_FOLDER/
quartus_pgm -m jtag -o "pvi;flash_image.jic"
```

   **Note:** If no previous design is running, or if the previous design has been running successfully, the MSEL does not need to be set to JTAG. The MSEL can remain set to QSPI, and you can still program the QSPI flash successfully. If a previous design is running, run `jtagconfig` to determine if the Intel Stratix 10 (SDM) is first or second in the JTAG scan chain. If it is second, change the above command to:
   
   ```bash
quartus_pgm -m jtag -o "pvi;flash_image.jic@2"
```

4. Set MSEL back to QSPI:
   
   SW2: 1: ON 2: OFF 3: OFF 4: ON
5. Connect a USB serial port with a console program like minicom, using 115200-8-N-1 settings, to the board.

6. Power cycle the board.

7. Stop at the U-Boot prompt by pressing any key during U-Boot countdown.

8. Run the bridge enable U-Boot command to open the bridges.

9. Run the run bootcmd to boot Linux.

10. Set "login" to root, when Linux asks. Linux does not request a password.

11. After Linux boots, set "login" to root and no password is required.

Poky (Yocto Project Reference Distro) 3.0.3 stratix10 /dev/ttyS0
stratix10 login: root
root@stratix10:~#

1.2.14. Testing the SPIM0

Test the spidev program by using it to read the flash id from the QSPI boot card connected to the FPGA flash slot for the following parts:

- Macronix MX25U128:
  
  ```
  root@stratix10:~# ./spidev_flash_test /dev/spidev0.0
  response(7): c2 25 38 c2 25 38
  ```

- Micron MT25QU01G:
  
  ```
  root@stratix10:~# ./spidev_flash_test /dev/spidev0.0
  response(7): 20 bb 21 10 44 00
  ```

- Micron MT25QU02G:
  
  ```
  root@stratix10:~# ./spidev_flash_test /dev/spidev0.0
  response(7): 20 bb 22 10 44 00
  ```

1.3. Document Revision History for the AN 923: Routing Intel Stratix 10 HPS Peripherals to FPGA Fabric

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020.10.12</td>
<td>20.3</td>
<td>Initial release</td>
</tr>
</tbody>
</table>