Scaler Router Board

C. Perkins

August 28, 2007

1 Introduction

The Scaler Router Board (RAT) contains 322 input pins, any of which can be routed to any of the 521 output pins (See Figure 1). An input pin may also be routed to as many output pins as necessary. Inputs are mostly in the form of 16-bit DSM inputs (on 34-pin connectors) and outputs are mostly in the form of 25-bit Scaler outputs (on 50-pin connectors; 1 pin pair is a clock signal). Four daughter boards perform this routing. A microcontroller talks to each of the four daughter boards and handles all communication with the user.

There are two communication paths to the outside world. The path to program the routing map and read back the current configuration is over ethernet. A serial port is used to watch debug messages from the microcontroller (this port is connected to a terminal server to allow remote monitoring). The only information that is sent over the serial connection to the microcontroller is IP information to configure the ethernet communications.

2 Design and Implementation

2.1 Daughterboard

Figure 2 shows a block diagram of the daughter board. Each daughter board has access to all of the motherboard's input pins and a subset of the board's output pins. The four daughter boards cover the complete set of output pins. The daughter board contains only a Xilinx Spartan-3 1500 BGA FPGA and footprints for two optional oscillators. In normal operation, these oscillators are not used and do not need to be loaded. Each of the four daughter board FPGAs is programmed from a separate PROM located on the mother board. In addition to power lines and FPGA configuration lines, each daughter board has external connections to 482 3.3V I/O lines. Not all of these I/O lines are used on any of the daughter boards but could be if the daughter board were reused on a different mother board.

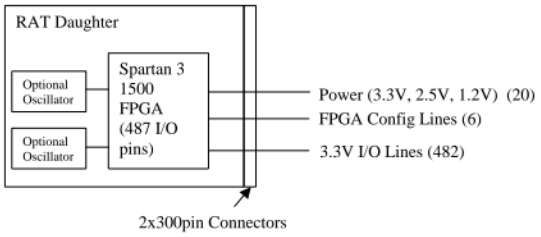

Figure 2: Scaler Router Daughterboard

There are two ways to configure the routing of inputs to outputs. The simplest way is to hard-code input lines to output lines directly in each daughter board VHDL. The software provided (described later) generates these VHDL files if this method is used. The drawbacks to this scheme are that each daughter board must be reprogrammed each time the routing map changes and that it takes a long time (~20 minutes) to program each PROM over ethernet.

A simpler method is to make each daughter board FPGA able to route any of its inputs to any of its outputs. The user communicates with the FPGA over ethernet to tell the FPGA how to route it. This scheme has the advantage that the VHDL code doesn't need to be reprogrammed when the routing is changed. To change the routing, the user must simply tell the FPGA the new map; the VHDL doesn't change. The drawback to this scheme is that daughter board #3 contains too many outputs to place and route this general design. Therefore, one of the scaler

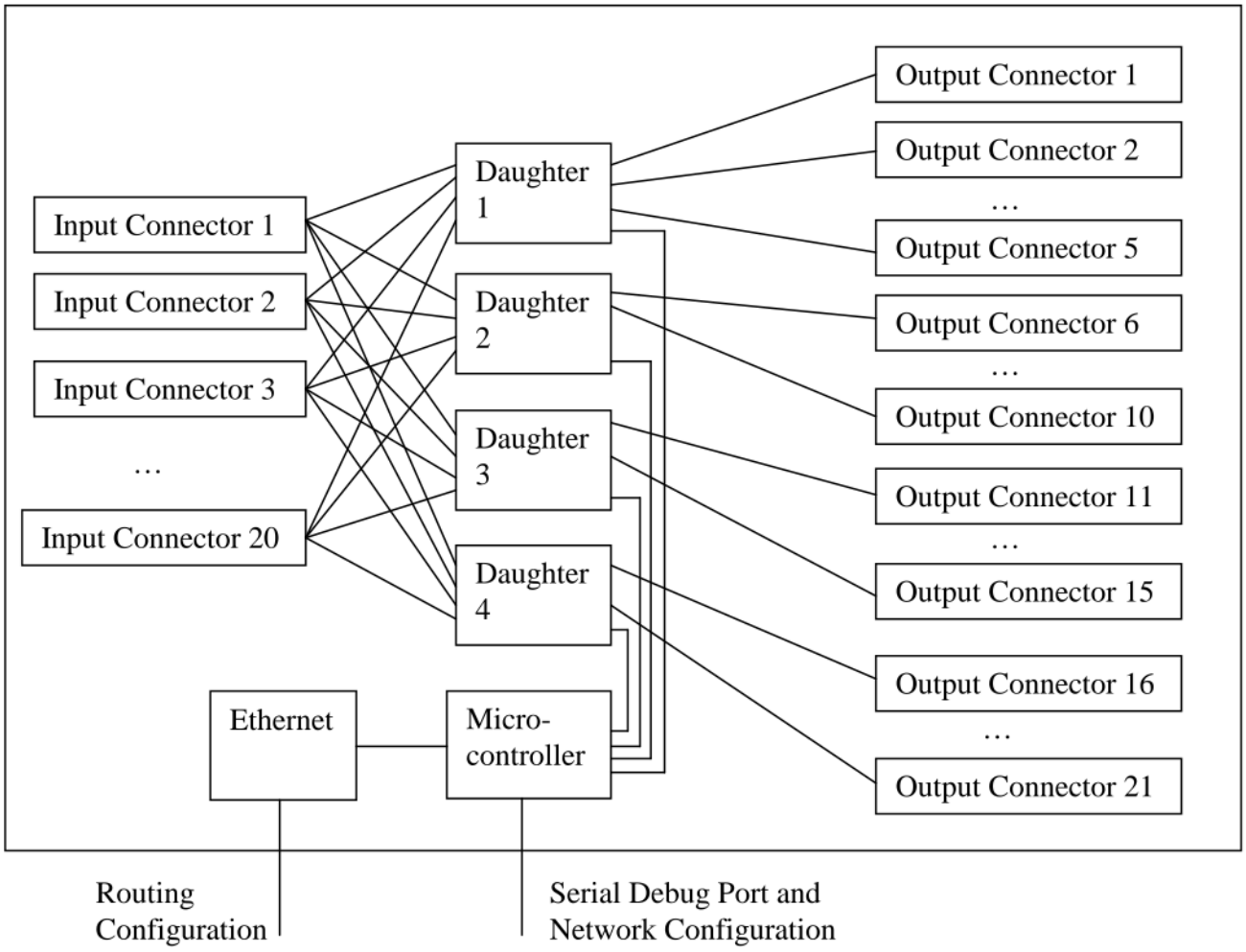

Figure 1: Scaler Router Motherboard

outputs (J36) must be omitted. If it is possible to leave this output open, these scheme is much easier and faster. If it is essential to use all scaler outputs, daughter board #3 must be programmed with the hardcoded VHDL but the other three daughter boards can be left with the general VHDL. Both schemes are described in further detail in the Software section.

2.2 Motherboard

Figure 1 shows a block diagram of the mother board. There are four daughter board slots per motherboard. The motherboard contains the input and output connectors as well as receiver and driver chips. The motherboard also handles all communication with the outside world.

2.2.1 Inputs

The following input connections are available:

- 16 34 pin DSM Inputs
- 1 10 pin RCC Input
- 1 40 pin GLINK Input
- 1 50 pin TCU Input
- 1 32 pin header

Input signals can be either PECL or ECL (check this) to accomodate DSM and TAC inputs. On the Scaler Router Board currently installed, each PECL input is terminated but if for some reason one desires that these signals are left unterminated, the resistors could be removed.

There are also 16 LEMO inputs available if TTL inputs are desired. These TTL inputs take the place of one of the DSM input connectors if used and can be toggled in two groups of 8 using jumpers J24 and J29. Note that both jumpers J24 and J29 must have a jumper on either pins 1 & 2 or pins 2 & 3 to choose PECL or TTL. If there is no jumper, neither format will work on that input.

2.2.2 Outputs

The following output connections are available:

- 17 50 pin Scaler Outputs
- 3 34 pin DSM Outputs
- 1 40 pin GLINK Output

All output signals are PECL. The last pin on each scaler output should be configured to be a copy of the RHIC clock in order for the scaler boards to function properly. Note that if the hardwired VHDL code is used for daughter board #3, an 18th Scaler output is available.

2.2.3 Communication and Programming

Communication is handled by an Atmel ATmega6450 microcontroller. There are two ways to talk to the microcontroller: over ethernet and through the serial port.

The onboard 10baseT ethernet controller (Cirrus CS8900a) can be connnected directly to a standard ethernet hub. The microcontroller contains a full TCP/IP stack that can be configured to use either DHCP or a static IP address. It is only programmed to respond to ICMP Ping requests and incoming connections to TCP port 9876.

TCP port 9876 can be used for two functions. First, it can be used to program the four PROMs that configure the FPGAs. Because of the speed of the microcontroller, this path is very slow. This is accomplished by sending "TDI" and "TMS" commands to the microcontroller using a specially designed protocol. The microcontroller has the appropriate connections to each of the four PROMs to program them independently.

The ethernet connection can also be used to configure the input-output mapping if the FPGAs are loaded with the "general" VHDL (as described above). Using a similar protocol to that mentioned above, the user sends commands to read or write a mapping to a designated daughter board. The microcontroller receives these commands and communicates directly with the appropriate FPGA, either reading or writing a mapping (using another specially designed protocol). There is also a command to reset all mappings. When write commands are sent, the mapping is also written to the microcontroller's EEPROM so that last used mappings remain in place even if power is removed.

See the accompanying software for examples

of how to communicate with the microcontroller over ethernet.

The second path to the microcontroller is through the serial port. This port is mostly used to monitor debug messages. To monitor this port, use a serial port set to 19200 baud, 8 Data bits, No Parity, 1 Stop bit, and No flow control. The Scaler Router is currently connected to port 4 on trgserv.trg.bnl.local which uses these settings.

This port is also used to configure the TCP/IP settings for the ethernet connection. When the board starts up, the user has 5 seconds on this port to press a key if they wish to change the ethernet settings. Pressing a key puts you into a very simple shell. Within this shell, press '?' to show a list of available commands. The commands available are:

- '?' Show commands
- 'p' Show current configuration
- 's' Choose between static IP and DHCP
- 'i' Set static IP address
- 'r' Set router address
- 'n' Set netmask
- 'q' Quit setup and boot

Because the shell is so simple, when entering an IP addres or netmask, you cannot use backspace. If you make a mistake, simply reenter the entire address. Be sure to print the current configuration and check the address, router, and netmask before you boot the controller. These settings are programmed into the microcontroller's EEPROM and therefore will remain in place even if power is removed.

The board is currently set to use a static IP address of 172.16.128.25 (on the trg.bnl.local network).

(optional JTAG port patch)

2.2.4 Microcontroller Programming

As you might guess, there is a good amount of code that is used to program the microcontroller. This code contains the TCP/IP stack and the code to interface with both the PROMs and the FPGAs. There should be no need to modify this code but in the case that one needs to, here is how its done.

This code is written mostly in C with some special ATmega6450 commands. To compile it, use AVR Studio 4 which is freely available. There is a project file included with the code that should be loaded in AVR Studio. To compile, choose "Build" from the "Build" menu, or press the button labelled "Build Active Configuration". The code should compile with no warnings and generates the file `scaler_router.hex`.

To program the ATmega6450, first set the jumpers on J62 and J63 for programming as described in the Jumpers section. Then connect a serial cable from you computer's serial port to the serial port on the mother board. To program the chip, you use a program called "PonyProg2000" with is also freely available. Start PonyProg and choose "AVR micro" for the device family and "ATmega64" for the device type. If this is a new board that has never been programmed before, you will need to program the "FUSE" bits first. (Set FUSE Bits).

After the "FUSE" bits are set, in the "File" menu, choose "Open Device File" and choose the previously generated `scaler_router.hex` file. Make sure that the Scaler Router Board is powered on and choose "Write All" from the "Command" menu. Choose "Yes" when it asks if you're sure you want to write to the device. It will then try to write to the device but come back with the message "Device Missing or Unknown Device". This is because PonyProg doesn't have the ATmega6450 chip. The ATmega64 has the same programming interface, though, so choose "Ignore" when this message comes up. It will then go through 2 write sequences (one for the FLASH memory and one for the EEPROM) and a verify sequence. After this, the board should be ready to use.

Note that after you reprogram the ATmega6450, you will need to change jumpers J62 and J63 back to serial communications and reload the IP address, router and netmask.

2.3 Hardware Modifications

The external oscillator connected to the CS8900a ethernet chip was connected wrong in the artwork. To fix this, lift pins 97 & 98 on the ethernet chip (U12) and connect the two pads of a XXMhz crystal across them (something like part number XXX). This patch will probably be very fragile and should be epoxied.

The only other error on the artwork is the 5A fuse where the power comes in. The board uses more than 5 Amps so this must be bypassed by soldering the input 5V cables to the other side of the fuse (labelled `v50_local` on the schematic).

(JTAG Ports. optional patch)

2.4 Software

The software to communicate with the board over ethernet (PROM programming and FPGA route mapping) was written for linux and can be used on any machine connected to the same network as the Scaler Router Board (trg.bnl.local in the current case). As a quick check, the board should respond to ordinary Ping requests.

First, you must setup the desired Input-Output mapping. This software resides in the `map_software` subdirectory. The files `InputMap.txt` and `OutputMap.txt` contain mappings that are set in the layout and should never be changed. When the Scaler Router Board is cabled, you need to make a dictionary file (see `dict.all_ins_test.txt` or `dict.txt` for examples). This file maps between connector numbers on the schematic and human readable names.

When the dictionary is ready, the next step is to make a file that maps input pins to output pins. See `ScalerRouterMap.all_ins_test.txt` or `ScalerRouterMap.txt` for examples.

Next, simply run:

```
gen_vhdl [InputMap] [OutputMap]
         [dictionary] [RouteFile]
```

where InputMap is `InputMap.txt`, OutputMap is `OutputMap.txt`, dictionary is your dictionary file, and RouteFile is your pin map. This should complete with no errors.

This code generates VHDL and UCF files for each daughter that can be synthesized, placed and routed into MCS files if the hardwired approach to routing is desired. The code also generates a file `ScalerRouter.cmd` that contains the commands needed to configure the routing map if the daughters are programmed with the "general" VHDL.

The software to program the PROMs or configure the FPGA routing map is in the `fpga_software` subdirectory.

```
fpga_config [hostname] [cmd_filename]
            [verify_after_program]
```

Configure the routing map on `hostname` using the file `cmd_filename` generated with `gen_vhdl`.

```
fpga_verify [hostname] [cmd_filename]
```

Check the currently configured routing map on `hostname` against the file `cmd_filename` (generated with `gen_vhdl`).

```
fpga_read_config [hostname]
```

Read back the currently configured routing map on `hostname`. The routing map is printed to the screen.

```
fpga_read [hostname] [daught_num]
          [output_num]
```

Read a single map entry on `hostname` corresponding to `daught_num`,`output_num`. The argument `output_num` is an index from 0 to the number of outputs on the daughter that is internal to the VHDL code. The argument `daught_num` should be between 1 and 4.

```
fpga_write [hostname] [daught_num]
           [output_num] [input_num]
```

Write a single map entry on `hostname` setting `daught_num`,`output_num` to `input_num`. The arguments `output_num` and `input_num` are indices from 0 to the number of outputs on the daughter card or the number of inputs (this index is internal to the VHDL code). The argument `daught_num` should be between 1 and 4.

```
prom_prog [hostname] [mcs_filename]
          [daught_num] [verify_after_program]
```

Program the PROM for daughter board `daught_num` on `hostname` using the MCS file `mcs_filename`. The argument `daught_num` should be between 1 and 4. After the PROM is programmed (and optionally verified), the corresponding FPGA will be loaded with the new code.

2.5 Jumpers

There are two sets of jumpers on the mother board. J24 and J29 allow the user to choose between DSM input J15 (PECL) and the 16 LEMO connectors (TTL). This input is split into an upper half and a lower half and the choice between PECL and TTL can be made independently for each half. Note that both jumpers J24 and J29 must have a jumper on either pins 1 & 2 or pins 2 & 3 to choose PECL or TTL. If there is no jumper, neither format will work on that input.

The second set of jumpers allow you to choose whether you are talking to the microprocessor or programming the microprocessor over the DSUB-9 serial port. To talk to the microprocessor, pins 1 & 2 should be connected (jumper on) and pins 3/4, 5/6, and 7/8 should be unconnected (jumper off)on header J62. Pins 2 & 3 should be connected with a jumper on header J63.

To program the microprocessor, pins 1 & 2 should be unconnected (jumper off) and pins 3 & 4, 5 & 6, and 7 & 8 should be connected (jumper on) on header J62. Pins 1 & 2 should be connected with a jumper on header J63.

2.6 Testing

First check that the ethernet configuration can be set over the serial port as described in the Communication and Programming section. After this is set, as a quick check of the ethernet connection, the board should respond to ordinary Ping requests. You should also see debug commands on the serial connection as the routing map is configured.

To check the routing map configuration, you can use the files `dict.all_ins_test.txt` and `ScalerRouterMap.all_ins_test.txt` to `gen_vhdl` as described in the Software section. Then use `fpga_config` to try configuring the routing map using the file `ScalerRouter.cmd` that was just generated. Be sure to set the `verify_after_programming` option so that the configuration is checked after it is loaded. If `fpga_config` verifies correctly, the communications and programming functions are all working.

To test the input and output channels, a DSM with a working upper output channel (bits 16-31) and working input channel 1 is needed. Configure the DSM to play from its output memory and record into its input memories. The file `DSM_OutIn_Loop_Test.dat` can be used for this purpose.

First test the outputs by generating a CMD file (using `gen_vhdl`) with the dictionary `dict.all_outs_test.txt` and map file `ScalerRouterMap.all_outs_test.txt`. Use `fpga_config` to configure the routing map. This configuration routes the inputs from 1 DSM input connector (J57) to all output pins.

Connect the upper DSM output channel (lower connector on the DSMI) to the Scaler Router input labelled J57. Connect DSM input channel 1 to the Scaler Router output labelled J48. By hand, put the DSM into run mode and then out of run mode.

Included in the software provided is a VXWorks library called `dsm_dump.o`. Load this library and run:

```
DSM_Dump(DSM_base_address,
         "rat_test.j57.j48.dat");
```

This will output a dump of the DSM memories into a file called `rat_test.j57.j48.dat`. Move this file to a linux machine to be checked later.

Now use the special cable that splits a Scaler output into two DSM connections. Connect the scaler end of the cable to the Scaler Router output labelled J44. Leave the DSM cable connected to DSM input channel 1. Remove the other end of this cable from Scaler Router output J48 and connect it to the lower bits of the scaler splitter cable.

Put the DSM into and out of run mode again and run:

```
DSM_Dump(DSM_base_address,
          "rat_test.j57.j44_lo.dat");
```

Move the DSM cable to the upper bits of the scaler splitter cable, put the DSM into and out of run mode, and run:

```
DSM_Dump(DSM_base_address,
          "rat_test.j57.j44_hi.dat");
```

Cycle through all the outputs on the Scaler Router board using this process, changing the output filename to `DSM_Dump` accordingly. Be sure to test both the high and low sections of the Scaler outputs as appropriate. There is also one 40pin output for which a special cable has also been made and a high and low section will need to be tested.

On the linux machine with the saved data files, run the supplied program:

```
check_dsm [datafile] 1
```

Since we only used the upper 16 bits of the DSM output, we only care about bits 16-31 in the output of this check. Ignore bits 0-15. The `check_dsm` program will report the percent that each bit was correct and the percent that each bit was a '1'. If the Scaler Router output tested was a 34pin output or the lower half of a scaler output, all 16 upper bits should be correct 100% of the time and should be '1' about 50% of the time. From the output, you should be able to tell if certain bits are stuck at '1' or '0'. If you are testing the upper bits of a Scaler output, you should only see the first nine bits (16-24) correct 100% of the time.

The second argument to `check_dsm` is an offset between output memory values and input memory values. This offset should be 1 but if you are having trouble getting 100% correct data, try changing this value to a few other values near 1.

To check the input channels, generate a CMD file (using `gen_vhdl`) with the dictionary `dict.all_ins_test.txt` and map file `ScalerRouterMap.all_ins_test.txt`. Use `fpga_config` to configure the routing map. This configuration routes the inputs from one input connector to one output connector. Therefore, both the input and output connected to the Scaler Router will have to be moved after each `DSM_Dump` call.

Input	Output
J56	J44
J57	J43
J51	J68
J55	J67
J59 – lo	J47
J59 – hi	J46
J61	J48 – hi
J53	J34
J52	J39
J69	J65
J50	J64
J54	J45
J70	J72
J49	J73
J58	J35
J66	J42
J71	J41
J30	J40
J15	J48 – lo
J75	J37
J60 – lo	J38 – lo
J60 – hi	J38 – hi

Figure 3: Input Output mapping for testing inputs

See Figure 3 for the input and output connectors to use for this test. For example, for the first test, connect the upper DSM output to Scaler Router input J56. Connect Scaler Router output J44 (lower half) to the DSM input channel 1. Put the DSM into and out of run mode and run:

```
DSM_Dump(DSM_base_address,
          "rat_test.j56.j44.dat");
```

Cycle through the list of inputs and outputs listed in Figure 3, saving the output file of `DSM_Dump` on a linux machine as before.

Run through the new data files with `check_dsm` checking that the appropriate bits are correct 100%.