Designing with the Programmable Serial Interface™ (PSI™) for High-Speed Solutions

Introduction

Warp™ Release 6.1 adds the ability to create designs for Programmable Serial Interface (PSI) devices, Cypress's family of programmable PHYs. PSI is a programmable PHY, ideal for use in high-speed backplane solutions. Designing with PSI is done with the same ease as with other Cypress programmable product families, such as the Delta39K or Ultra37000. For more information about PSI, please refer to the following Cypress website:

http://www.cypress.com/psi.

Similar to programmable logic, there are a large number of ways to layout a design on the PSI device. Understanding device specific architecture is imperative to making effective design implementation choices for optimized performance. This application note presents the design requirements for PSI together with multiple strategies for laying out a design to achieve optimum speed, while efficiently allocating the programmable logic resources.

The PSI2G100(S), now available with *Warp* Release 6.1, is a high-speed PSI that operates at a narrow-band frequency of 2.5 Gbps. As such, logic programmed in the PSI must be able to meet the timing requirements for the serial link interface. To guarantee the proper operation of the PSI device, two requirements need to be observed:

- **All high-speed datapath input signals to and output signals from the SERDES block must be registered at a standard datapath cell.**
- **The worst case register-to-register delay (T_{scs}) for any PSI logic must be less than the target operating frequency period of that clock domain.**

A list of the recommended strategies applicable to high-speed designs is shown below. One or more of these techniques may be applied to critical path signals to improve the speed performance of a design.

- **Register all device input and output signals at an I/O cell.**
- **When passing a signal asynchronously across the PSI, choose I/O cells that are in the same I/O block or choose an output cell located in the I/O block directly across the device from the input cell I/O block.**
- **Add synchronous pipelined stages into the design as needed.**
- **Partition the logic so that there is no more than one pass of logic between registers.**
- **For system-clock-to-system-clock paths between pipelined stages, avoid the use of Horizontal-to-Vertical (H-to-V) or V-to-H switches.**
- **When using a cluster or dual port memory, register the data at the input to, and at the output from memory.**

Design Requirements

Registering High-Speed Datapath Signals Interfacing with the SERDES

All high-speed datapath signals interfacing with the SERDES must be registered at standard datapath cells. This is particularly important for the SERDES transmit and receive data.

This may be done by placing the lines of VHDL or Verilog code assigning the values of these signals in one or more `process` or `always` blocks, synchronized to the SERDES transmit or receive clock. This is imperative in achieving the strict timing requirements of the PSI.

The user may have to instruct the *Warp* fitter to register the SERDES input signals at the standard datapath cells by using the `input_reg` attribute and giving it the value `ioreg_iocell` in the design control file. The same may be done for SERDES output signals by using the `output_reg` attribute and assigning it to `ioreg_duplicate` in the design control file. *Listing 1* shows an example.

Listing 1. Registering inputs and outputs using synthesis directives.

```
Attribute input_reg of <rxdata>: signal is
 ioreg_iocell;
Attribute output_reg of <txdata>: signal is
 ioreg_duplicate;
```

Checking Worst Case Datapath T_{scs}

The worst case register-to-register delay (T_{scs}) for any clock domain in the PSI logic must be less than the target operating frequency period of that clock domain. For example, in the PSI2G100(S), the operating frequency range of the 16-bit SERDES interface is 154.5 MHz to 156.5 MHz. This means that the worst case T_{scs} between any combination of registers in this clock domain should be less than 6.47 ns to 6.38 ns respectively. If the delay is greater than the period of the target frequency, the affected signals will fail to meet the timing for the next register in the datapath.

The worst-case T_{scs} can be determined easily in the *Warp* Release 6.1 software. This number is available after synthesis of a design and can be found in two places.

One place to view this information is in the *Warp* Release 6.1 Architecture Explorer. From the Architecture Explorer menubar, select **Tools** > **Static Timing Analyzer** to open the Static Timing Analyzer dialog box. A shortcut is also available by clicking on the Static Timing Analyzer button on the Architecture Explorer toolbar.

Static Timing Analyzer button.

Figure 1 shows the Static Timing Analyzer dialog box. There are four tabs available near the bottom of the dialog box. Select the **tSCS** tab as shown in *Figure 1*.

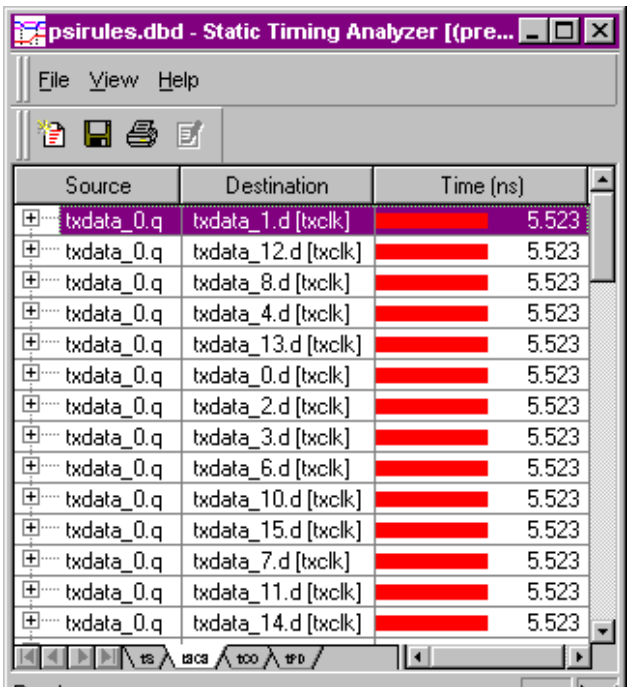

Figure 1. Static Timing Analyzer dialog box.

All the register-to-register paths are listed in the dialog, as well as the timing delay between the two registers. By default, these are listed in decreasing order, showing the worst case T_{SCS} at the top of the list. Check that all registers along the critical datapath have a delay of less than the target operating frequency period. If the T_{SCS} delay exceeds the operating period, a set of recommendations are made in the next section of this application note that assist in making the design meet timing requirements.

The other place that contains the timing information is the design report file. In the *Warp* Release 6.1 main window, choose the **Output Files View** tab from the **Project** window. Locate and expand the *<top_level_file>.rpt* file by clicking on the "+" sign preceding the filename. View the timing information in the report file by double-clicking on **Timing**. The text window will show the section of the report file containing the timing information of the various timing parameters. Scroll down and look for **Register-to-register times (tscs)**. The timing information from the timing analyzer is shown in text format in the post-fit report file in *Figure 3*.

Design Recommendations

Registering Input and Output Signals to the Device

It is good practice to register all input and output signals at the I/O cells. Registering input signals synchronizes the inputs and makes it easier to meet the setup time requirements for the next register in the design. Registering the output signals at the I/O cells ensures that all output signals are synchronous and also achieves the minimum clock-to-output time from the PSI output signal to the next device. This is illustrated in *Figure 2*.

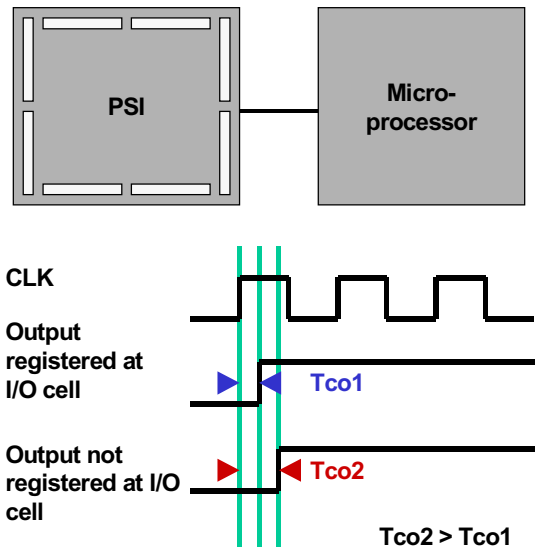

Figure 2. Registering an output at the I/O cell minimizes the clock-to-output time from the PSI output signal to the next device [1,2]

Notes:
1. Tco1 is the t_{IOCO} parameter in the PSI data sheet.
2. Tco2 is the t_{UCCO} parameter in the PSI data sheet.

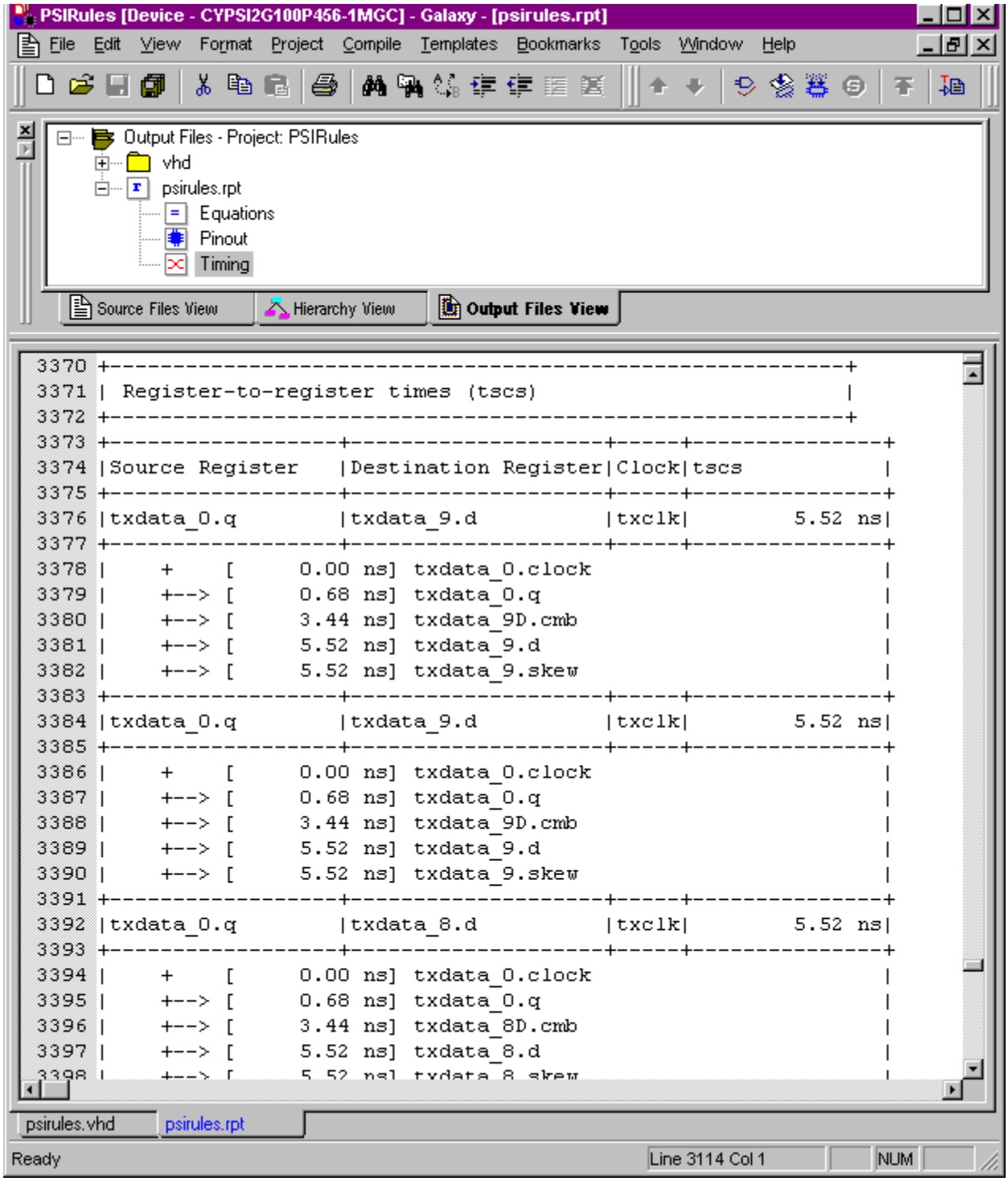

Figure 3. Viewing the Timing Information in the Design Report File.

Similar to the standard datapath cells, the *Warp* fitter may also be instructed to register the input signal to an I/O cell by assigning the attribute `input_reg` to the value `ioreg_iocell` in the design control file. Again, the same may be done for output signals by assigning the attribute `output_reg` to the value `ioreg_duplicate` in the design control file. An example was shown in *Listing 1*.

Passing a Signal Asynchronously across the PSI

Under certain circumstances, a designer may want to pass a signal across the PSI without undergoing any logical processing. This is commonly done with PSI control and status signals. For example, a designer may pass a SERDES status signal directly to the device I/O cell for output.

When performing such an operation, choose I/O cells that are in the same I/O block, or choose an output cell located in the I/O block directly across from the input cell I/O block. These recommended I/O block pairs with their respective dedicated channel connections are shown in color and also labeled with the same letter in *Figure 4*. I/O cells belonging to the same I/O block can be directly linked with very little delay. I/O blocks directly across from each other are connected by dedicated horizontal or vertical channels that add minimal delay to the signal propagation.

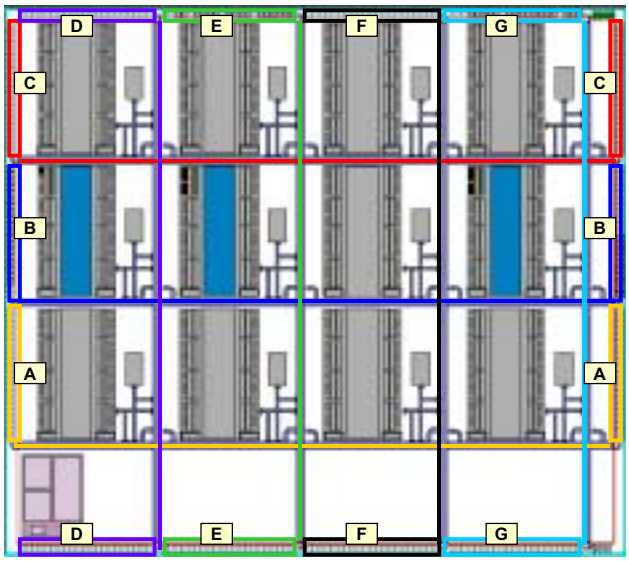

Figure 4. PSI2G100(S) architecture showing I/O block groups to be used for passing signals asynchronously to achieve minimum delay.

Listing 2 shows an example of how to lock a signal to an I/O pin using the synthesis directives in the design control file.

Listing 2. Assigning pins using synthesis directives.

```
Attribute pin_numbers of <input_signal>:signal
 is “D3”;
Attribute pin_numbers of <output_signal>:signal
 is “N2”;
```

The string (entered in quotes) describes the name of the I/O pin to be assigned in the PSI architecture. The names of each pin can be viewed through the Architecture Explorer in *Warp* Release 6.1 by zooming in sufficiently to see the pin name at each I/O or standard datapath cell.

Introducing Synchronous Pipeline Stages

A large combinatorial datapath operation executed over one clock cycle may be broken up into smaller operations that are executed over multiple shorter clock cycles. In portions of the design where large computations are being performed, introducing intermediate registers increases the throughput of data by increasing the maximum speed of the design. This can be accomplished by placing lines of VHDL (or Verilog) combinational source code into one or more new processes (or `always` blocks).

The *Warp* fitter may also be instructed to factor sub-expressions into a node so that the user can control and optimize the physical routing path of a signal. This is achieved by using the attribute `synthesis_off` and giving it the value `true` in the design control file. *Listing 3* shows an example.

Listing 3. Factoring sub-expressions using synthesis directives.

```
Attribute synthesis_off of <signa11>: signal is
 true;
Attribute synthesis_off of <signal2>: signal is
 true;
```

In this example, the assignments for *signal1* and *signal2* respectively are factored and may be routed to various registers.

Placing Pipeline Registers

When pipelining registers in your design, it is important to ensure that the signals meet the necessary setup times for the next stage of the process. The longer the path to the next register, the longer the delay. For high-speed designs, it is best to place input connection registers along the same horizontal or vertical channels that are used to enter the device. This optimizes the speed of the design by taking a shorter path and avoiding the use of an H-to-V (or V-to-H) switch, which adds delay to the signal propagation.

Figure 5 shows an example of which logic block clusters are best used for pipelining an input signal coming from an I/O cell on the bottom left corner of the device.

Figure 6 shows another example of which logic block clusters are best used for pipelining a different input signal. In this second example, the input signal is coming from an I/O cell on the top right corner of the device.

There are 12 logic block clusters in the PSI2G100(S) architecture. Each logic block cluster is further divided into eight logic array blocks labeled A to H. *Figure 7* shows how the logic block clusters and logic array blocks are labeled in the PSI architecture.

The *Warp* fitter may be instructed to force a signal to be registered in a particular logic array block by using synthesis directives. *Listing 4* shows an example of locking a signal to a logic array block E in cluster C(0,2) without specifying a particular macrocell.

Listing 4. Assigning a signal to a logic array block using synthesis directives.

```
Attribute lab_force of <signal3>: signal is
 “C(0,2)E”;
```

The *Warp* fitter may also be instructed to force a signal to be registered in a particular macrocell in a specific logic array block. There are sixteen macrocells (0 to 15) in each logic array block. *Listing 5* shows an example of locking a signal to macrocell E0 in cluster C(0,2).

Listing 5. Assigning a signal to a macrocell using synthesis directives.

```
Attribute macrocell_loc of <signal4>: signal is
 “C(0,2)E0”;
```

Using Cluster or Channel Memories

When using a cluster or dual port memory, data should be registered during both input to and output from memory. Registering the data reduces propagation delay associated with asynchronous read or write processes.

Registering memory inputs and outputs are done during the LPM module instantiation in the HDL description of a design. First, assign the value **LPM_REGISTERED** to the input data, output data, and address control generics present in the `generic map` portion of the LPM instantiation. Secondly, specify a read and write clock in the `port map` section of the LPM module instantiation. *Listing 6* shows an example of the instantiation of a single-port RAM LPM module with registered inputs and outputs.

Listing 6. Instantiation of a single-port RAM LPM with registered inputs and outputs.

```
U1: mram_io   -- Single-port RAM with bidirec
              -- tional data in and data out.
   generic map(
       lpm_width => 8,  -- 8-bit word
       lpm_widthad => 4,-- 4-bit address
       lpm_numwords => 0,-- optional
       lpm_indata => LPM_REGISTERED,
       lpm_address_control => LPM_REGISTERED,
       lpm_outdata => LPM_REGISTERED,
       lpm_file => "", -- optional
       lpm_hint => speed -- optional
       )

   port map(
       dio => dio,    -- bidirectional data
       address => address,
       inclock => wclk, -- write clock
       outclock => rclk,-- read clock
       memenab => one,  -- optional
       outenab => zero, -- optional
       we => one, -- optional
       outreg_ar => zero -- optional
       );
```

Conclusions

Two requirements must be met to run a PSI design at speed.

- **All high-speed datapath input signals from and output signals to the SERDES block must be registered at standard datapath cells.**
- **The worst case register-to-register delay for any PSI logic must be less than the period of the target frequency of operation of that clock domain.**

If this is not achieved during the initial synthesis, each of the forementioned techniques can be employed across an entire design or for individual critical path signals to aid in meeting design speed requirements.

A PSI tutorial is available in chapter 5 of the Getting Started Manual in *Warp* Release 6.1. For additional information on using synthesis directives and working with a control file, please refer to *Chapter 4 – Synthesis Directives* in the *Warp* User's Guide, accessible from the *Warp* Help menu. For a description of the LPM modules available with *Warp*, please refer to *Chapter 5 – LPM* also in the *Warp* User's Guide. For design assistance or specific technical concerns, please contact your local FAE or Cypress applications.

cyapps@cypress.com

(408) 943-2821 extn.2.

Cypress PSI devices provide enormous flexibility in programmable logic while meeting the requirements of high-speed applications. They are an ideal solution for SONET OC-48, InfiniBand™, and system backplanes in high-speed networking and communications systems.

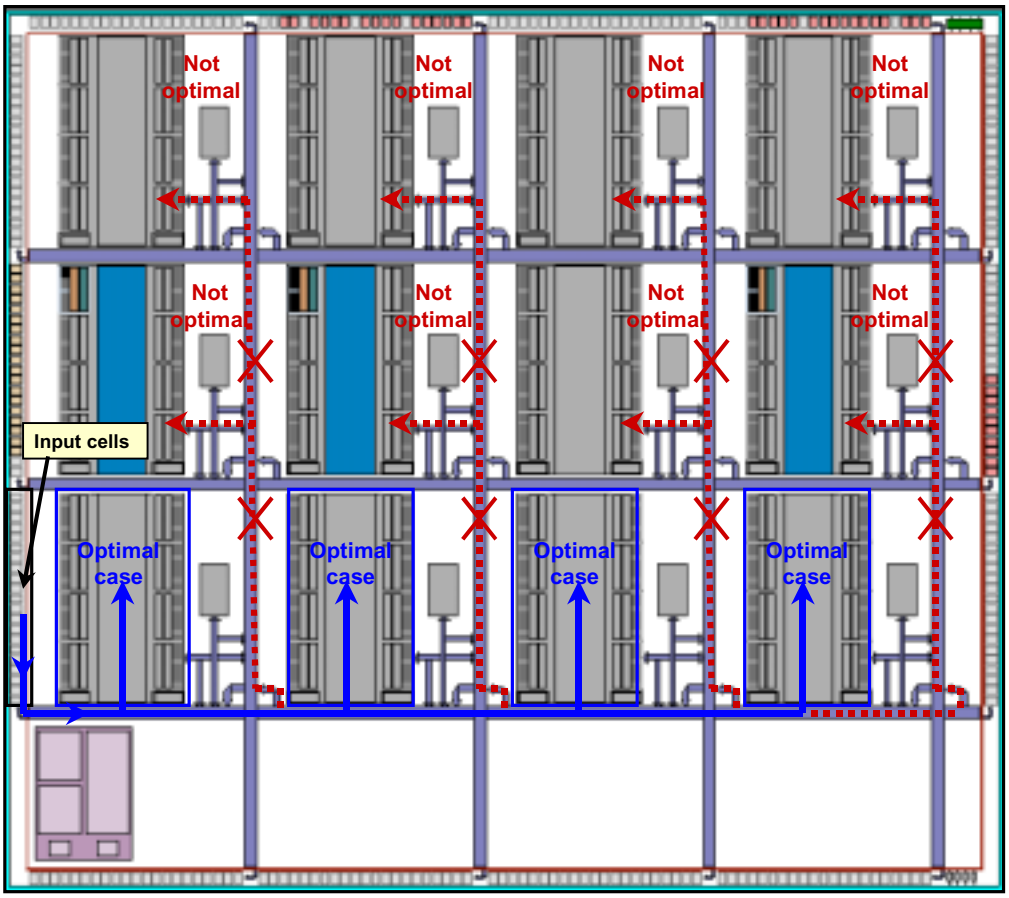

Figure 5. Example showing the best logic block cluster placement of pipelined registers when the input cell is located at the bottom left corner of the device.

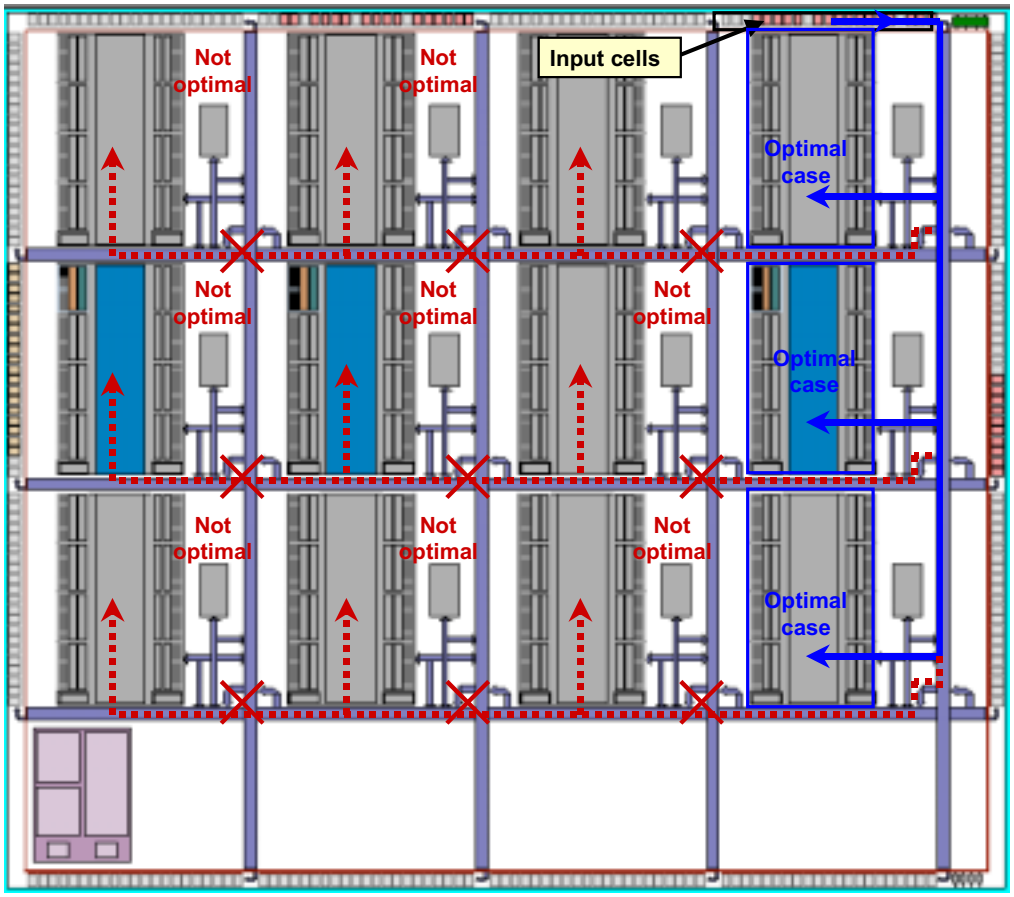

Figure 6. Example showing best logic block cluster placement of pipelined registers when the input cell is located at the top right corner of the device.

Figure 7. PSI2G100(S) logic block clusters and logic array block labels.