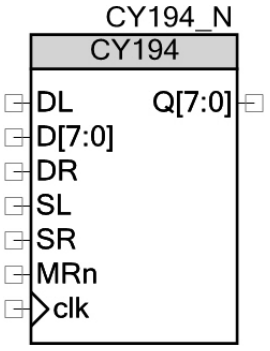
8-Bit Bidirectional Universal Shift Register

CY194 v1.0



Features

- Shift-left and shift-right capability
- Synchronous parallel and serial data transfer
- Easily expanded for both serial and parallel operation.
- Asynchronous master reset
- Hold (“do nothing”) mode
- Simple to deconstruct

General Description

This shift register is based on the 74HC194. It is a bidirectional 8-bit shift register with fully synchronous serial and parallel data entry. The registers are fully synchronous. As shown in the function table, the data can be shifted from left to right, right to left, or parallel data can entered, loading all 8 bits of the data simultaneously. When both SL and SR are LOW existing data is held in a hold (”do nothing”) mode. When both are HIGH a synchronous parallel load of the input data is performed. The first and last stages provide D-type serial data input (DT, DR) to allow multistage transfers without interfering with parallel load operation. When held LOW the asynchronous master reset (MRn) overrides all other conditions and forces the Q output LOW.

This component was built as a teaching tool. This classic component’s operation is well understood and this datasheet’s function is to help understand the component was built through its deconstruction.

Pin Description

Pin	Type	Function
DL	input	serial data input (shift left)
D[7:0]	Inputs	parallel data input
DR	Inputs	serial data input (shift right)
MRn	input	asynchronous master reset (active low)
Clk	input	clock input
Q[7:0]	outputs	parallel outputs

Function Table

Operating Modes	Inputs				Output							
	MRn	clk	SL	SR	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
reset	0	x	x	x	0	0	0	0	0	0	0	Q0
hold (do nothing)	1	↑	0	0	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
serial shift right	1	↑	0	1	DR	Q7	Q6	Q5	Q4	Q3	Q2	Q1
serial shift left	1	↑	1	0	Q6	Q5	Q4	Q3	Q2	Q1	Q0	DL
parallel load	1	↑	1	1	D7	D6	D5	D4	D3	D2	D1	D0

Deconstructing the Component

This component has connections to the Parallel Out (PO) and Parallel In (PI) interface. If you do not understand how they operate then first review the CY161 component. This component is a good example of how to perform both a shift right and shift left in a single datapath.

The datapaths were designed to allow either left or right shift functions with serial inputs and outputs. There is a workaround by having the datapath perform shift right instruction and implementing the shift left by adding a register with itself. It also shows how to route a serial input into a register.

For this shifter there are five different operations.

- Reset
- Hold (do nothing)
- Serial Shift Right
- Serial Shift Left
- Parallel load

While the reset operation will be handled with datapath reset circuitry, the other four operations need to be implemented with datapath instructions.

Using PSoC Creator, open the CY194 example project to see the project schematic (*TopDesign.cysch*). It has a CY194 component connected to input switches, output LEDs, and a clock.

In the Workspace Explorer, click the **Components** tab. Then, right-click on the project and select **Import Component**. Navigate to where the CYCC_SimpleComponentLibrary project is, and select the CYCC_CY194_v1_0 component. Click **OK** and the following files are shown for the component:

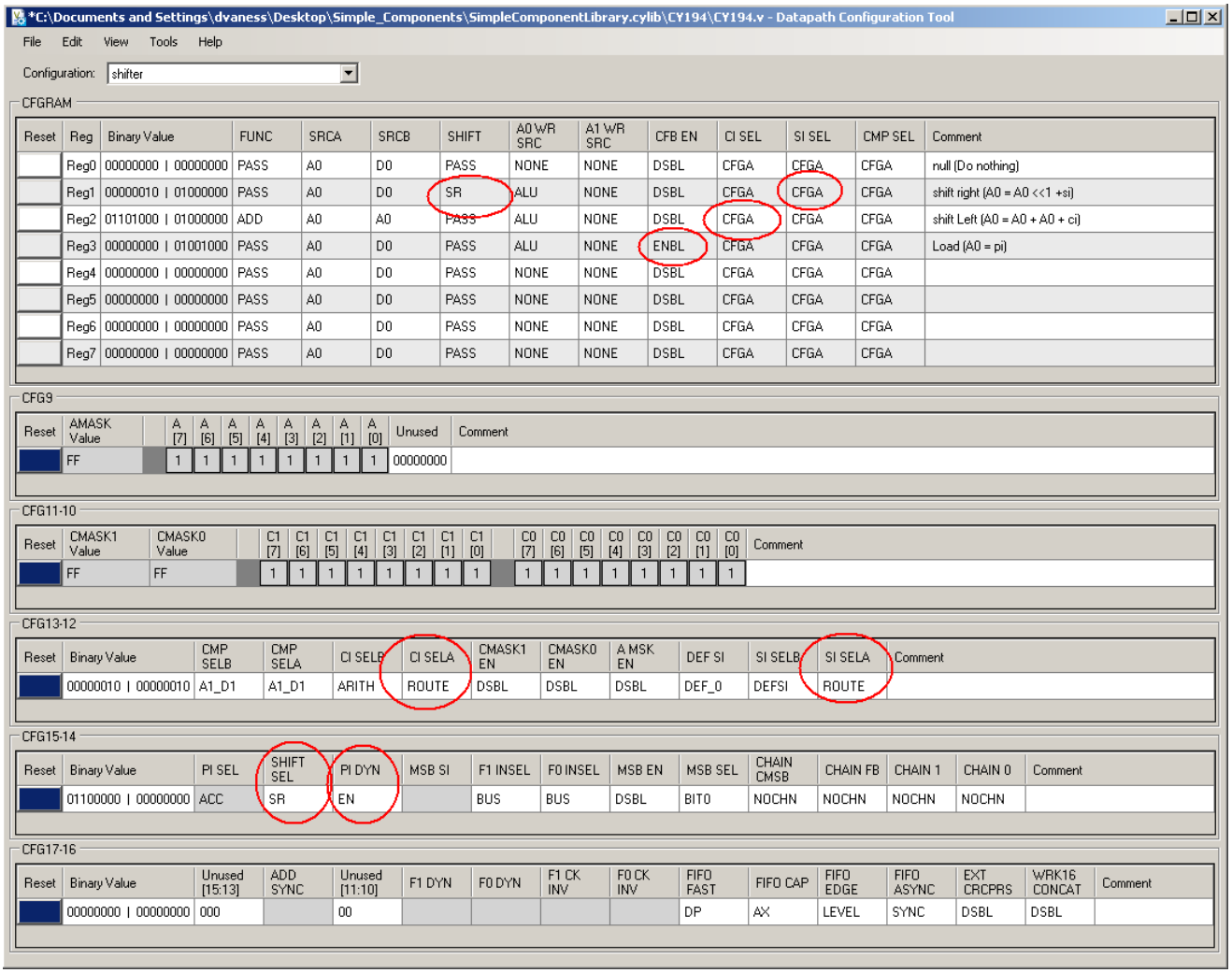
- Symbol file (cysym)

- Datasheet (pdf)
- Verilog File (v)

Open the symbol file to find a symbol with six inputs and one output. It looks like the symbol shown on the first page. There are no additional symbol parameters.

Open the Verilog file and notice that at lines 26 – 33, these definitions were passed to this Verilog file when it was created. The first 20 lines of this header list register usage and the datapath instruction definitions.

There is a need for intermediate signals and this is handled in lines 38 - 39. What follows is the datapath module definition. It was created and inserted by the Datapath Configuration Tool. This information is backward compatible so opening up the datapath configuration tool for this Verilog file results in the following.



The PI_DYN bit has been enabled so the CFB_EN bit of each instruction controls the ASRC input to the ALU. The shift select is set for right shift. SI SELA is set for ROUTE to connect the

serial input to the shift register. CI SELA is also set for ROUTE to connect the carry input to the shift register.

- For the null instructions A0 is passed through the ALU but not fed to any register. It does nothing.
- For the shift right instruction, A0 passed through the ALU and is right shifted, along with DR before being fed back to A0.
- For the shift left instruction, A0 is added with itself and the carry input. The result is fed back into A0.
- For the parallel load instruction the parallel input is passed through the ALU and fed back to A0.

When saved, this tool inserts the updated configuration data back into the Verilog file along with an instance of the datapath interface. Just pass the correct signals in to and out of it and you are done. This shifter parameter list may look ominous but only a few lines to change.

```
/* input          */ .reset(reset),             // Reset signal to datapath
/* input          */ .clk(clk),                 // Clock input to datapath
/* input [02:00]  */ .cs_addr({1'b0, SL, SR }), // Instruction control to datapath
/* input          */ .route_si(DR),             // Data for right shift
/* input          */ .route_ci(DL),             // Data for left shift
/* input [07:00]  */ .pi(D[7:0]),               // Parallel input to datapath
/* output [07:00] */ .po(Q[7:0])                // Datapath to parallel output
```

These parameters do the following:

- Connects a reset signal to the datapath reset circuitry
- Connects a clock signal to the datapath clock circuitry
- Control for datapath instruction processing
- Routes the shift right serial input signal into the shifter
- Routes the shift left serial input through the carry input
- Connects the parallel input into the datapath
- Connects the datapath to the parallel output

Support

PSoC Creator Community Components are developed and supported by the Cypress Developer Community. Go to www.cypress.com/CommunityComponents to discuss this and other Community Components.