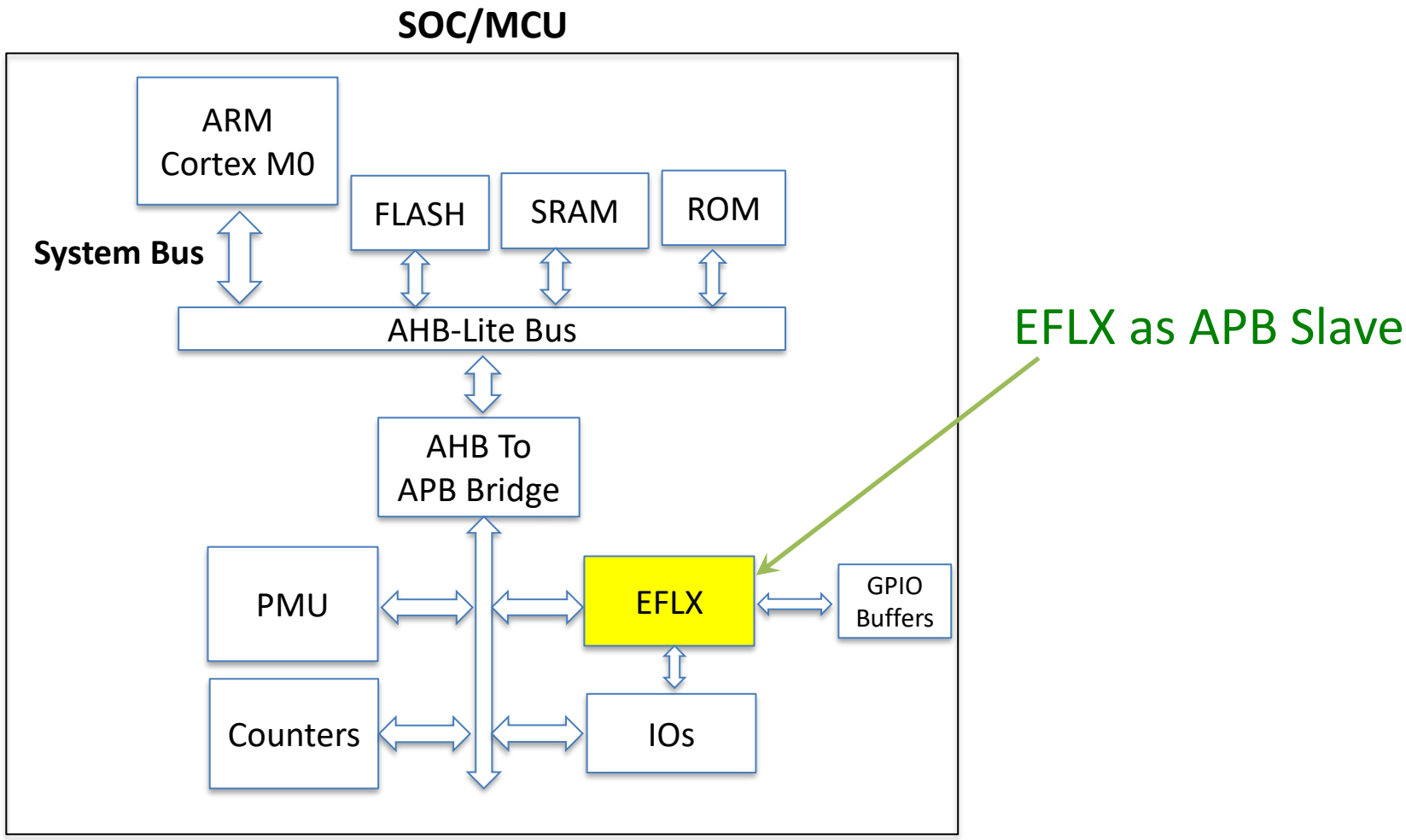
EFLX™ Customizable I/O on APB Bus

Updated March 7, 2023

Introduction

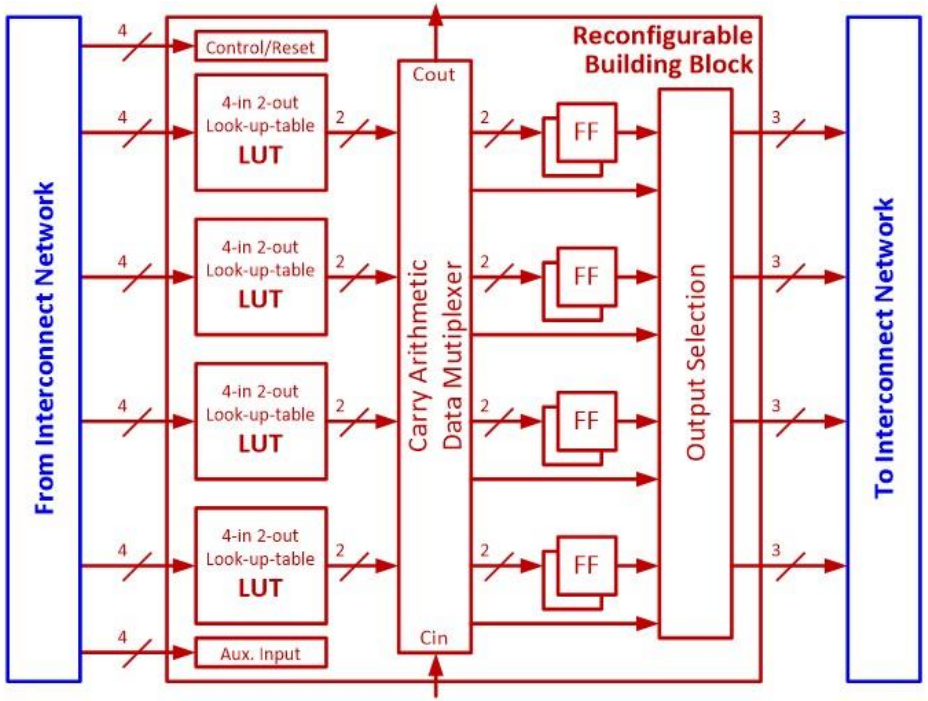
- EFLX1K or EFLX4K core can be added to SOCs/MCUs as an Intellectual Property (IP) block.
- Most SOC/MCU buses support the Advanced Microcontroller Bus Architecture (AMBA) buses.
- EFLX1K/4K cores support general purpose I/O functions and are not hard coded to any specific interface bus.
- EFLX1K/4K can be configured with the ability to interface to one of the AMBA buses and to become a building block in the SOC/MCU and make it easier to integrate without impacting other functionality.
- AMBA buses all use either input or output signals and not bidirectional signaling which allows EFLX1K/4K to interface directly onto these buses.

EFLX Integrated into ARM Cortex M0 MCU



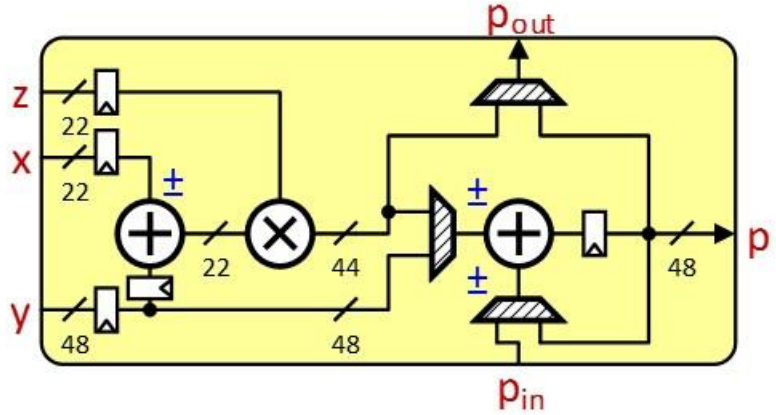
flexlogix
Technologies, Inc.

EFLX1K/4K Logic RBBs & DSPs



Logic RBB

- 4 Dual 4-input LUTs
- 8 Flip Flops
- Carry chain
- Selectable clocks

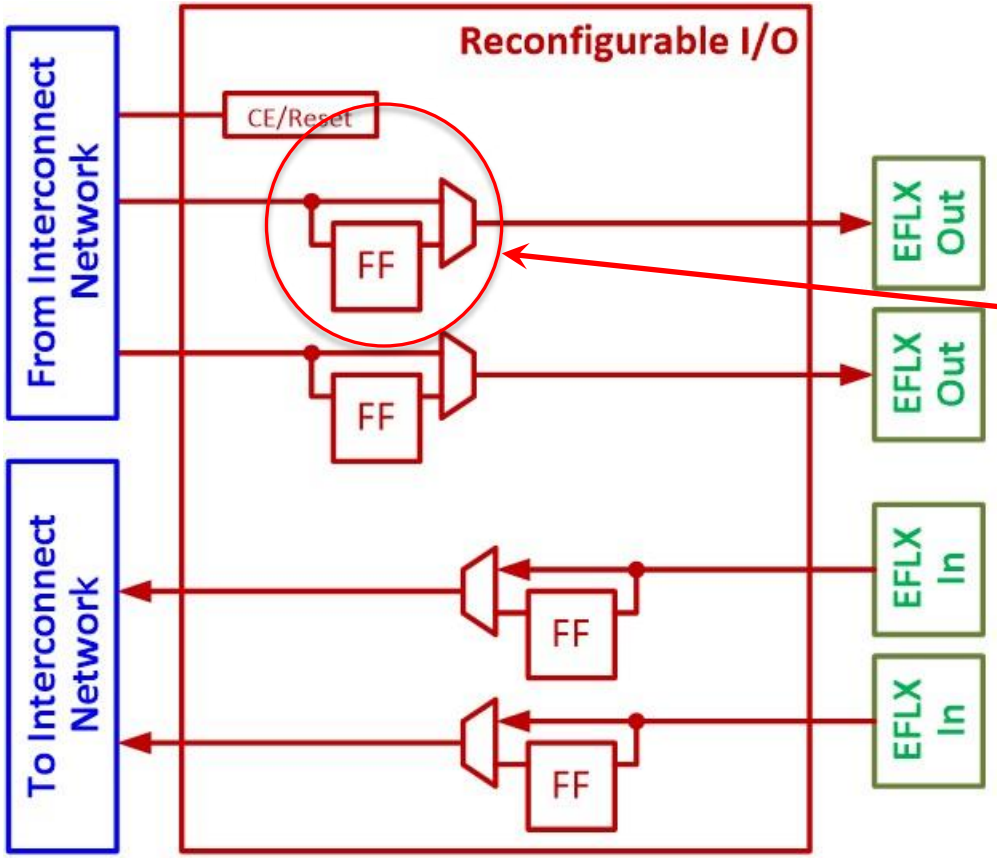


DSP RBB

- MAC
- 22-bit pre adder
- 22x22 multiplier
- 48-bit post adder (accumulator)

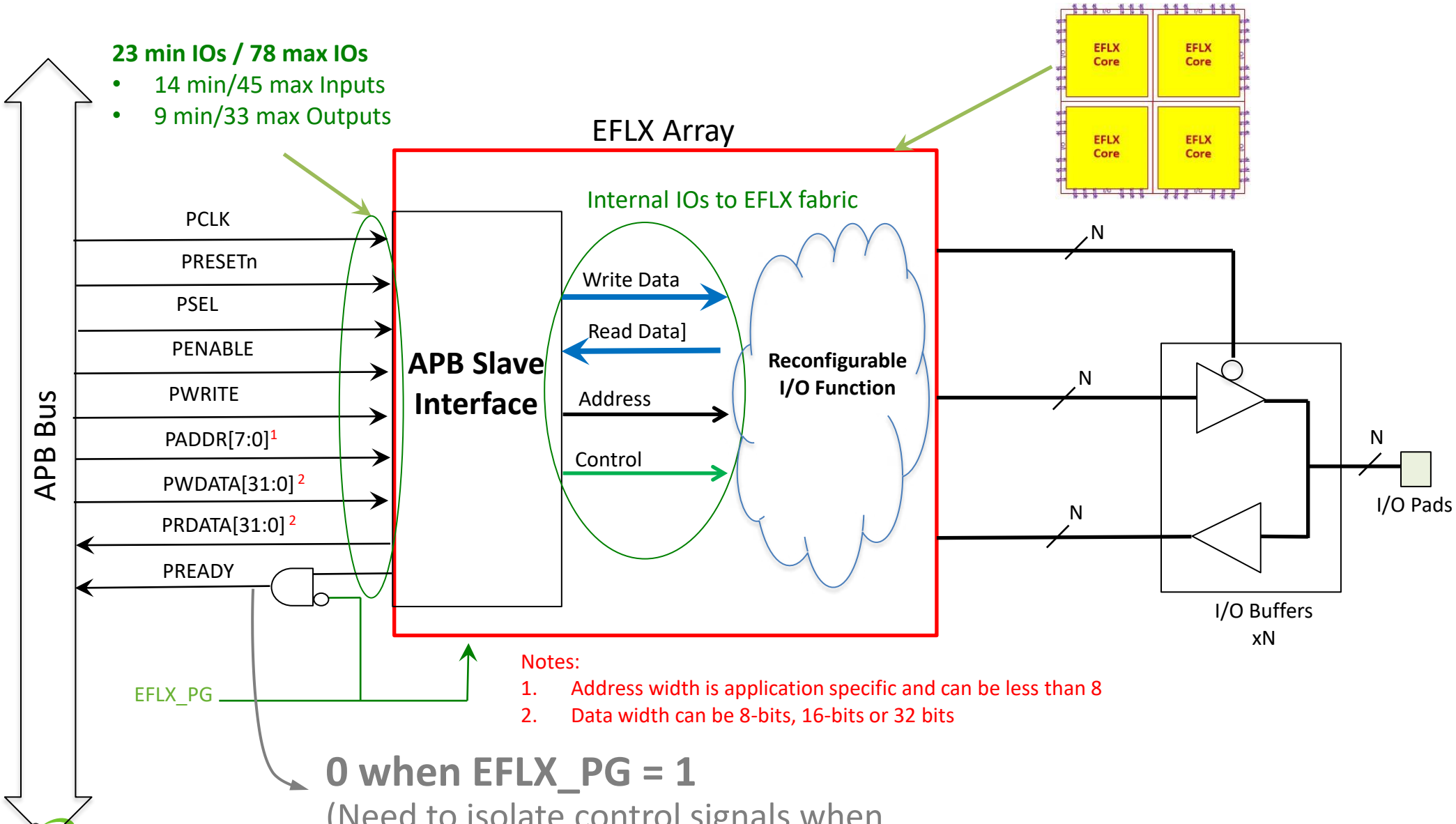
flexlogix Technologies, Inc.

EFLX1K/4K I/O RBB



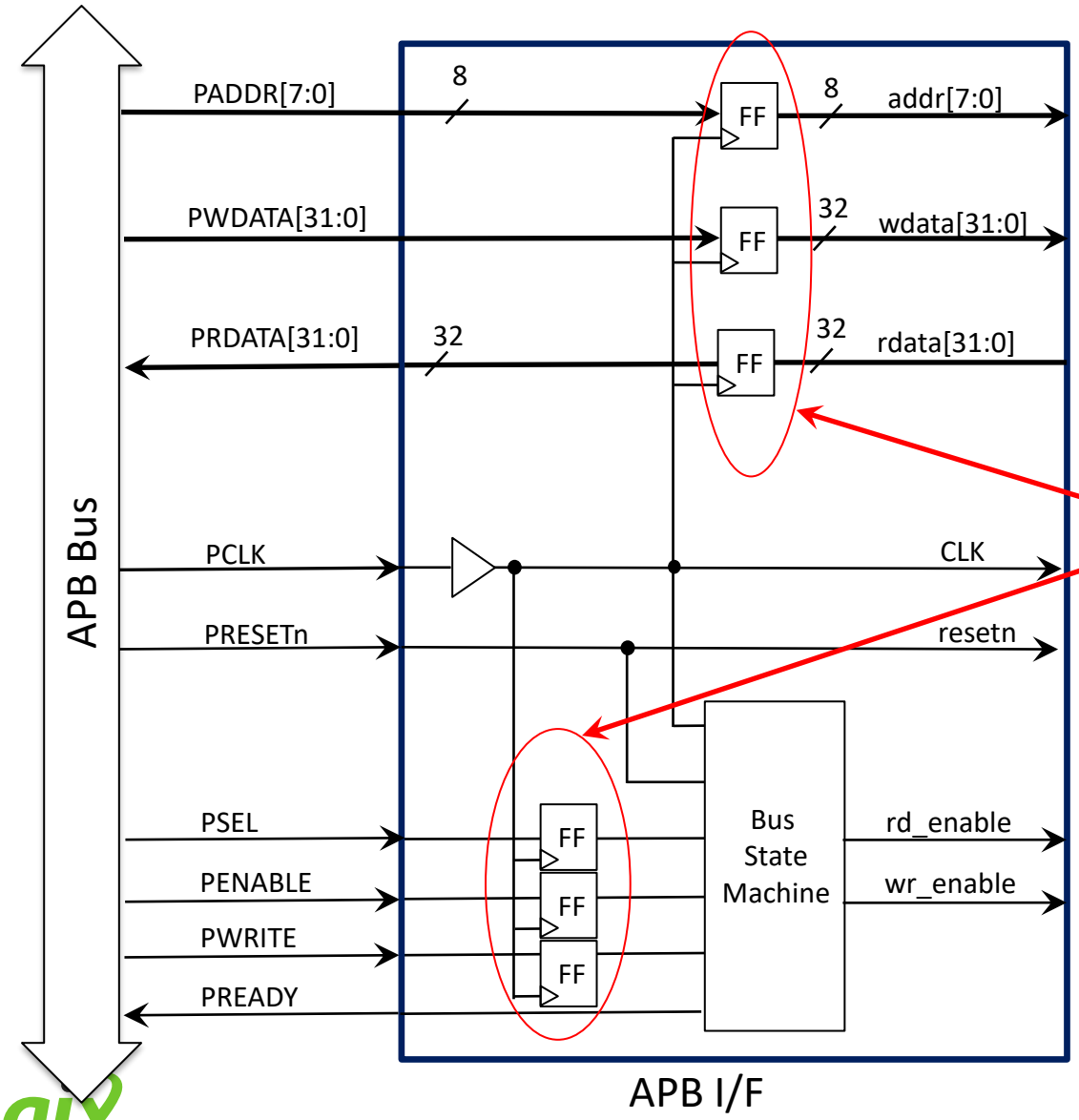
Inputs and outputs can be pass-through or flopped

EFLX with APB Slave Interface



flexlogix Technologies, Inc.

APB Slave Interface Details



- APB I/F part of programmable fabric
- Only takes 2 logic RBBs and 6 LUTs

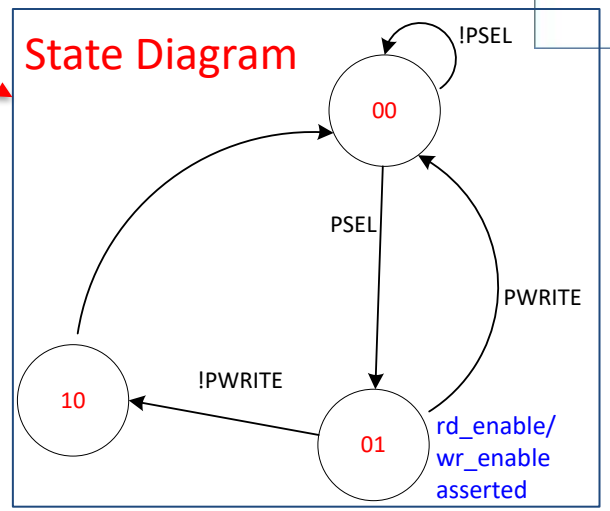
All I/Os are pipelined between the APB bus and EFLX fabric for timing isolation:

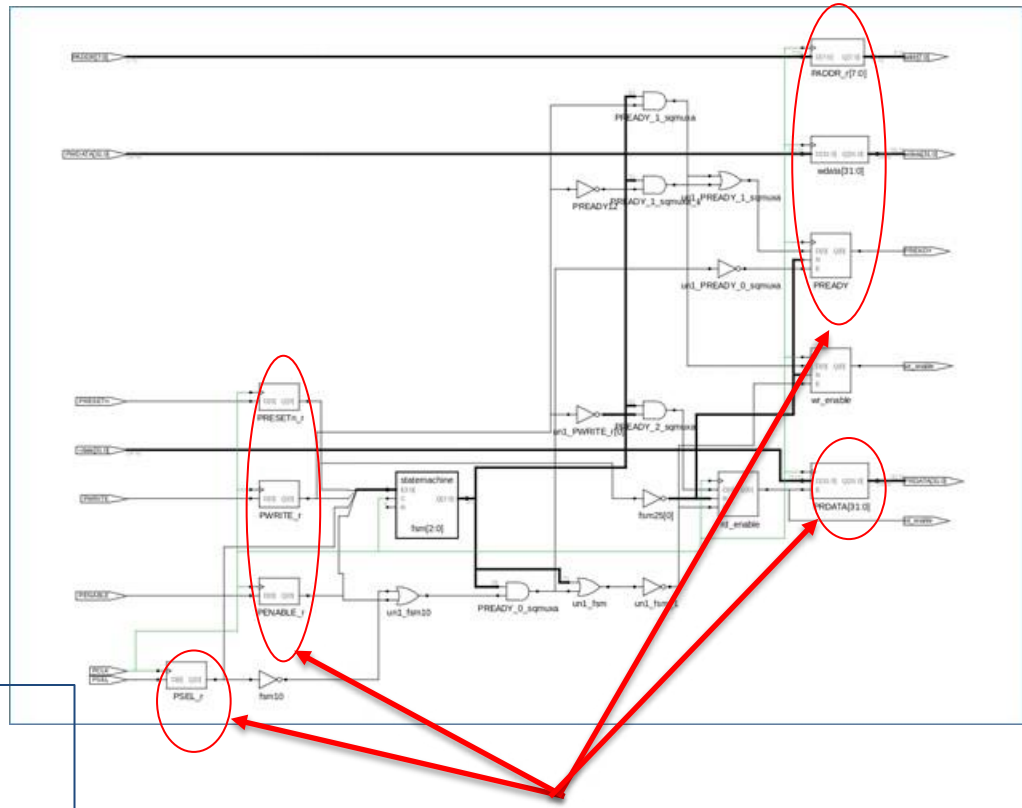
flexlogix Technologies, Inc.

APB Slave Interface Verilog Code

- Writes are 1 wait states due to pipelining
- Reads are 2 wait state2 due to pipelining

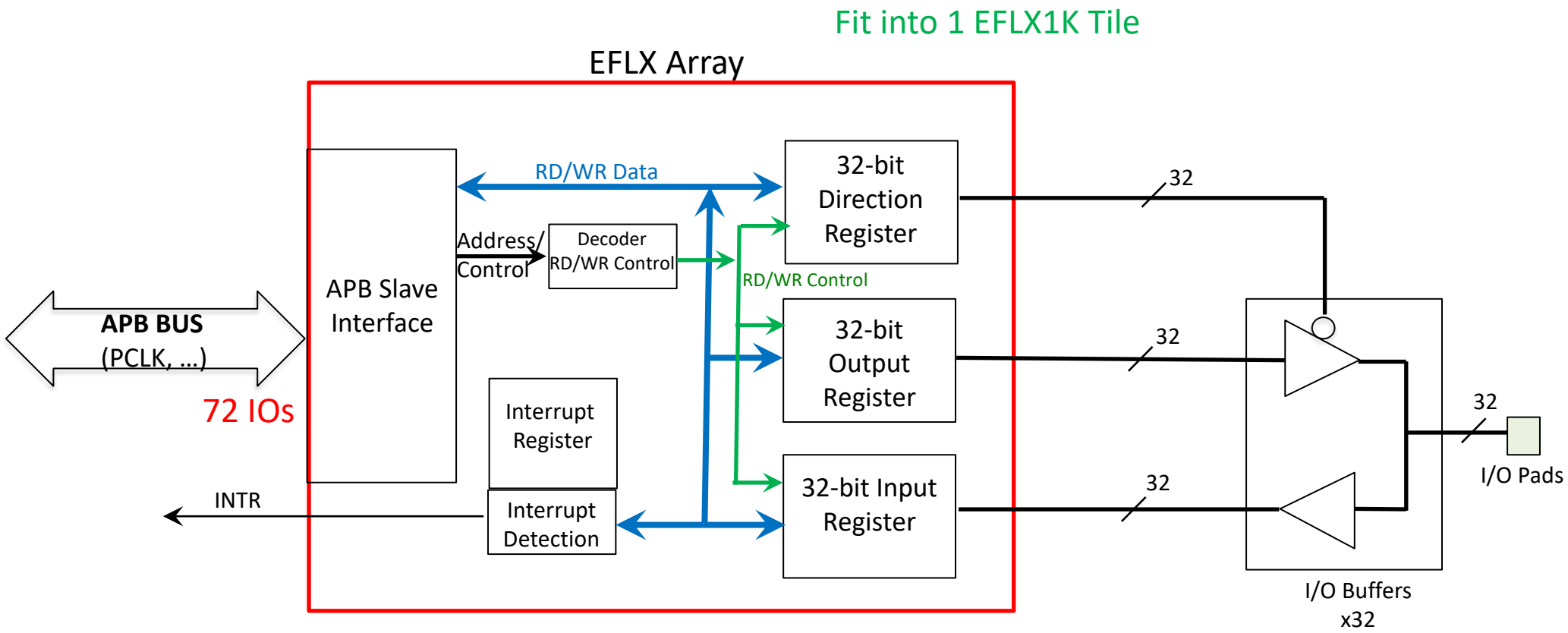
```
always @(posedge PCLK)
begin
  if (PRESETn_r == 0)
        begin
                fsm <= 0;
                wr_enable <=0;
                rd_enable<=0;
                PREADY<=0;
        end

else begin

  case (fsm)
        2'b00 : begin
                        if (~PSEL_r)
                                begin
                                        fsm <= 2'b00;
                                end
                        else
                                begin

                                        if (PWRITE_r==1 & PENABLE_r==0)
                                                begin
                                                        rd_enable <= 1'b0;
                                                        wr_enable <= 1'b1;
                                                        PREADY <= 1'b1;
                                                        fsm<=2'b01;
                                                end
                                        else if (PWRITE_r==0 & PENABLE_r==0)
                                                begin
                                                        rd_enable <= 1'b1;
                                                        wr_enable <= 1'b0;
                                                        PREADY <= 1'b0;
                                                        fsm<=2'b01;
                                                end
                                        end
                                end
        2'b01 :
                                begin
                                if (PWRITE_r)
                                        begin
                                                rd_enable<=1'b0;
                                                wr_enable <=1'b0;
                                                PREADY <=1'b0;
                                                fsm<=2'b00;
                                        end

                                else
                                        begin
                                                rd_enable <= 1'b0;
                                                wr_enable <= 1'b0;
                                                PREADY <= 1'b1;
                                        fsm <= 2'b10;
                                end
                                end
        2'b10: begin
                                        fsm <=2'b00;
                                        PREADY<=1'b0;
                        end
    endcase
end
end
```





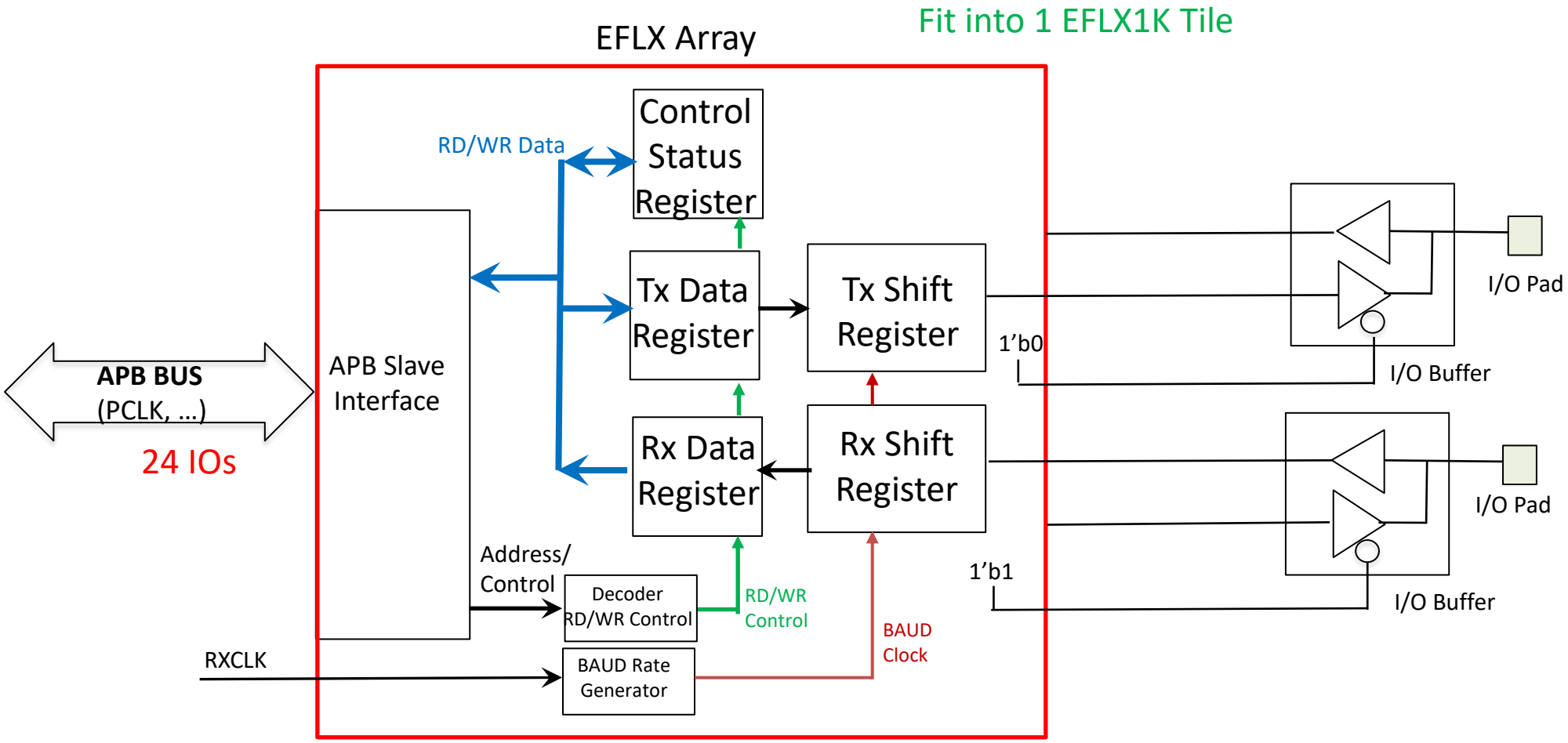
All I/Os to and from EFLX array are registered

APB 32-Bit GPIO Port



flexlogix Technologies, Inc.

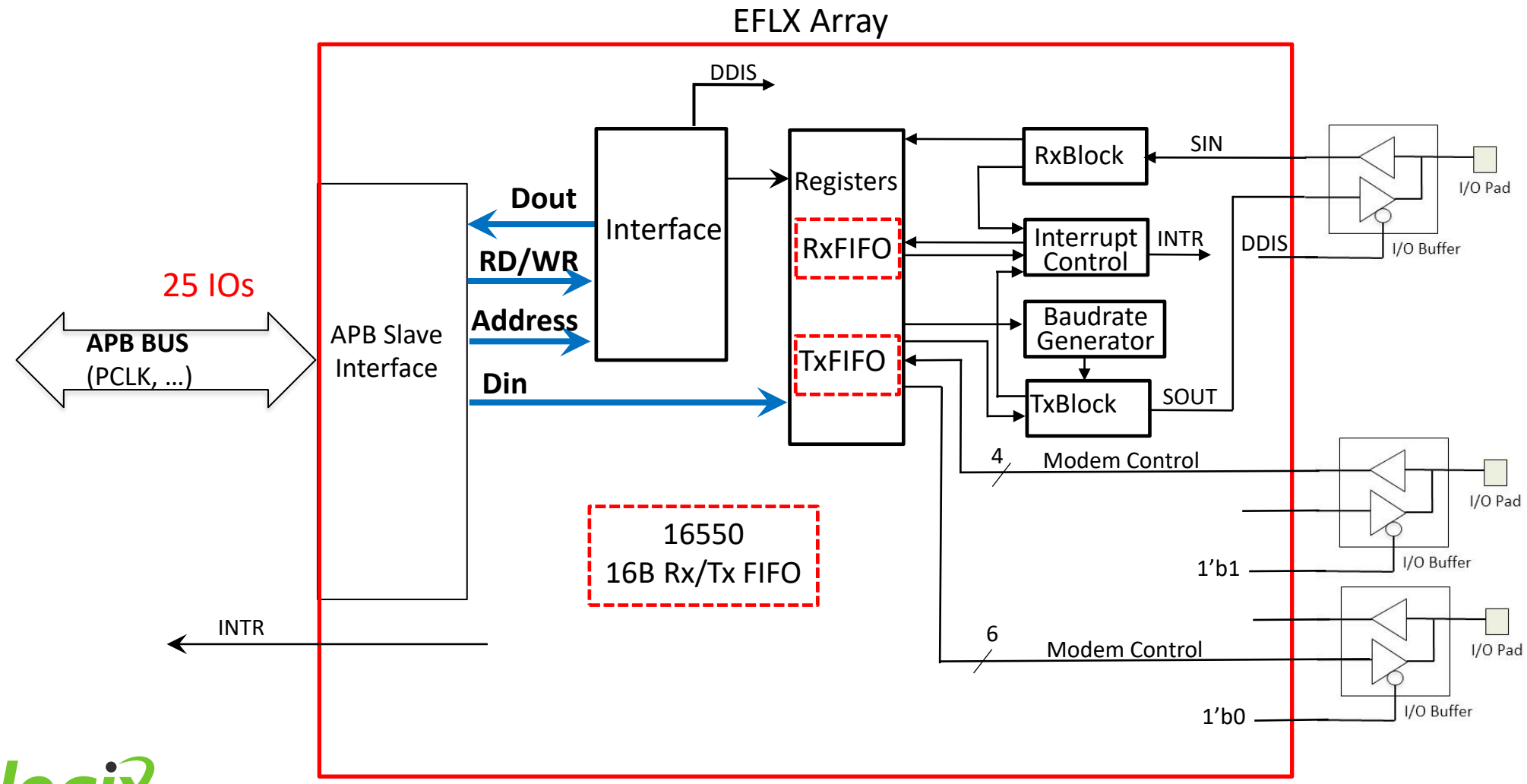
APB Simple UART



flexlogix Technologies, Inc.

APB 16550 UART

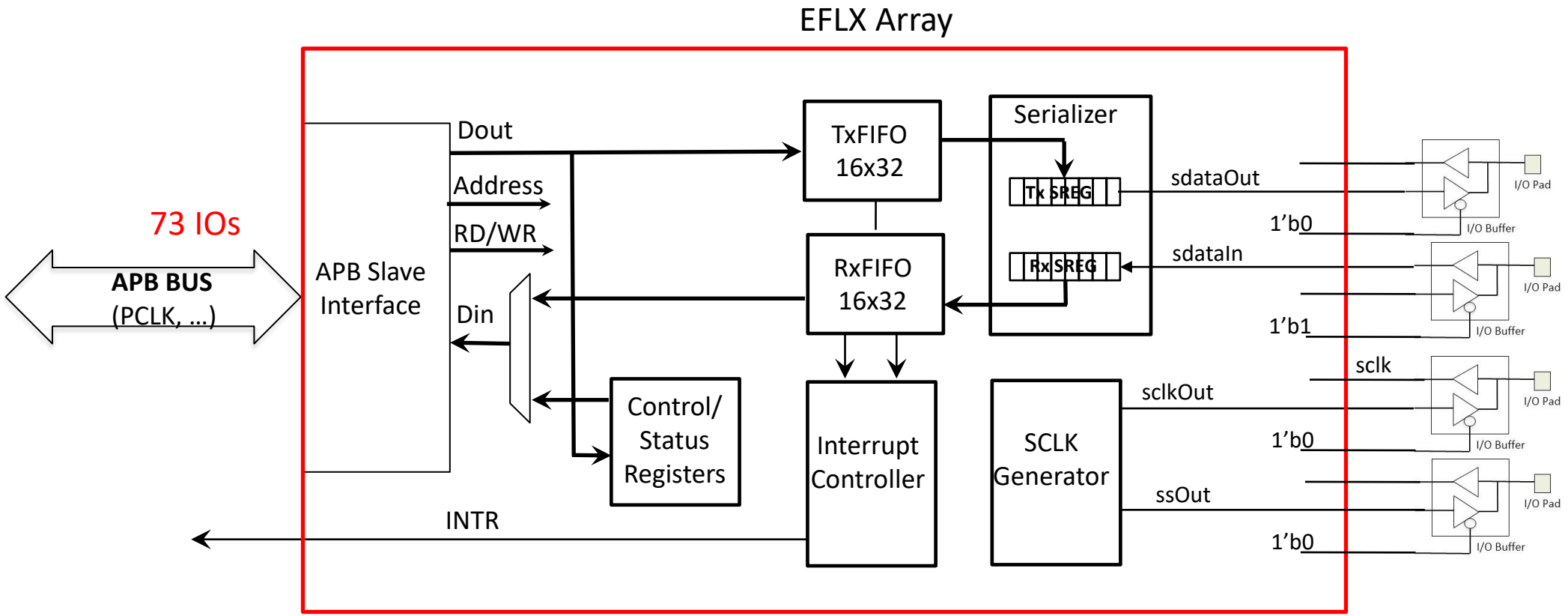
fit into 1 EFLX4K Tile



flexlogix
Technologies, Inc.

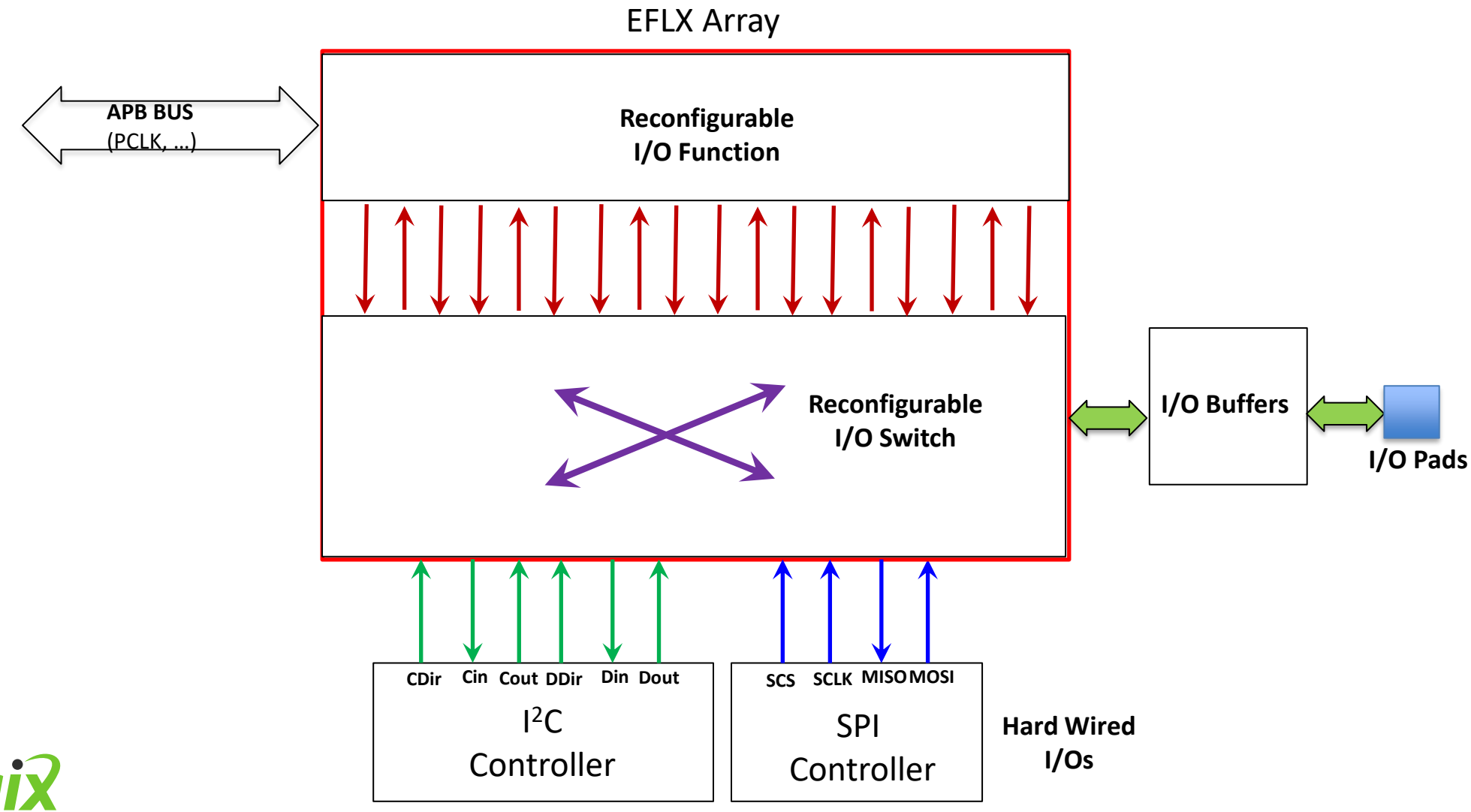
APB SPI Master

Fit into 1 EFLX4K Tile

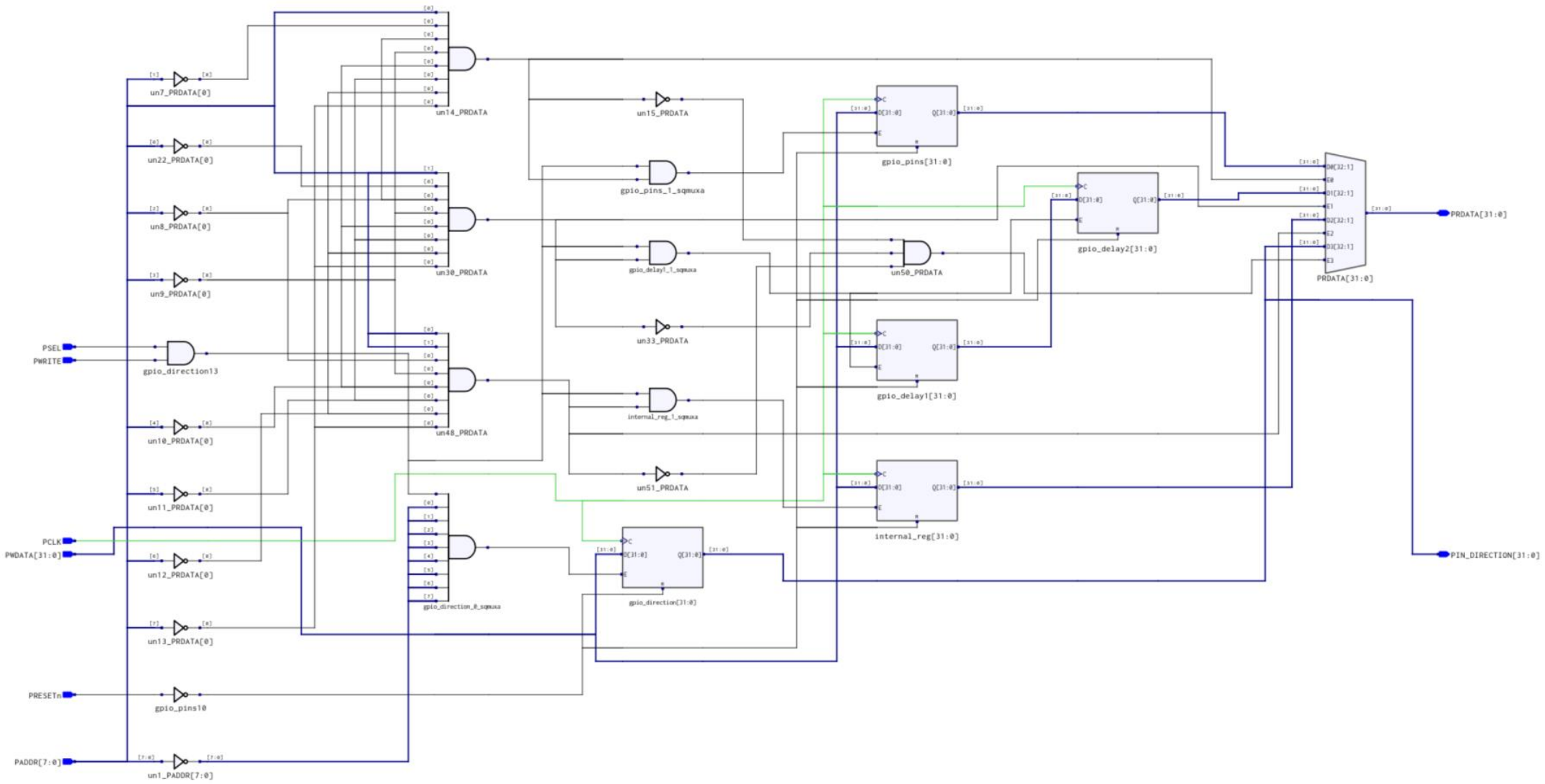


flexlogix Technologies, Inc.

EFLX as an I/O Switch

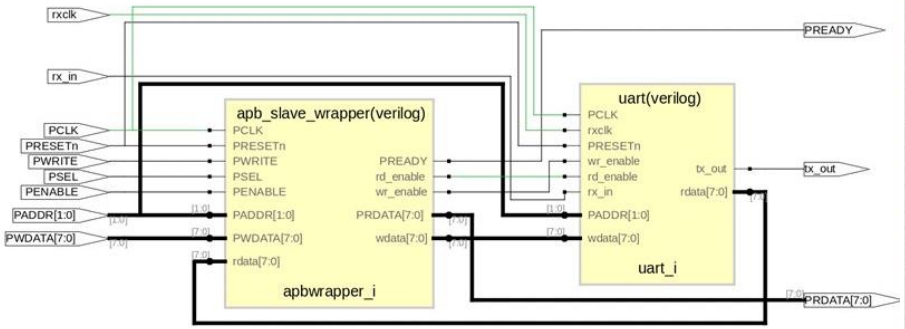


APB 32-bit GPIO Example



flexlogix Technologies, Inc.

APB Simple UART Example




```
end


// UART RX Logic
always @ (posedge rxclk /* or negedge PRESETn*/)
if (PRESETn==0)
        begin
                rx_reg        <= 0;
                rx_data       <= 0;
                rx_sample_cnt <= 0;
                rx_cnt        <= 0;
                rx_frame_err  <= 0;
                rx_over_run   <= 0;
                rx_empty      <= 1;
                rx_d1         <= 1;
                rx_d2         <= 1;
                rx_busy       <= 0;
        end
else
        begin
// Synchronize the asynch signal
        rx_d1 <= rx_in;
        rx_d2 <= rx_d1;
// Uload the rx data
        if (uld_rx_data)
                begin
                rx_data  <= rx_reg;
                rx_empty <= 1;
                end
// Receive data only when rx is enabled
                if (rx_enable)
                begin
// Check if just received start of frame
                        if ( ! rx_busy &&  ! rx_d2)
                        begin
                        rx_busy       <= 1;
                        rx_sample_cnt <= 1;
                        rx_cnt        <= 0;
                        end
// Start of frame detected, Proceed with rest of data
                        if (rx_busy)
                        begin
                        rx_sample_cnt <= rx_sample_cnt + 1;
// Logic to sample at middle of data
                                if (rx_sample_cnt == 7)
                                begin
                                        if ((rx_d2 == 1) && (rx_cnt == 0))
                                        begin
                                                rx_busy <= 0;
                                        end
                                        else
                                        begin
                                        rx_cnt <= rx_cnt + 1;
// Start storing the rx data
                                                if (rx_cnt > 0 && rx_cnt < 9)
                                                begin
                                                rx_reg[7] <=rx_d2;
                                                rx_reg[6] <=rx_reg[7];
                                                rx_reg[5] <=rx_reg[6];
                                                rx_reg[4] <=rx_reg[5];
                                                rx_reg[3] <=rx_reg[4];
                                                rx_reg[2] <=rx_reg[3];
                                                rx_reg[1] <=rx_reg[2];
                                                rx_reg[0] <=rx_reg[1];

                                                end
                                                if (rx_cnt == 9)
                                                begin
                                                rx_busy <= 0;
// Check if last rx data was not unloaded,
                                                   rx_over_run  <= (rx_empty) ? 0 : 1;
                                                   end
                                        end
                                end
                        end
                end
        end
        if ( ! rx_enable)
        begin
        rx_busy <= 0;
        end
    end
```

```
// UART TX Logic
always @ (posedge rxclk /*baudcnt[3] or negedge PRESETn*/)
if (PRESETn==0)
        begin
                tx_reg          <= 0;
                tx_empty        <= 1;
                tx_over_run <= 0;
                tx_out      <= 1;
                tx_cnt      <= 0;
                baudcnt         <= 0;
        end

else if (baudcnt<4'hf)
baudcnt<=baudcnt+1;

else  begin
        baudcnt<=baudcnt+1;
        if (ld_tx_data) begin
                if ( ! tx_empty) begin
                   tx_over_run <= 0;
                end else begin
                   tx_reg   <= tx_data;
                   tx_empty <= 0;
                end
        end
        if (tx_enable &&  ! tx_empty) begin
                tx_cnt <= tx_cnt + 1;
                if (tx_cnt == 0) begin
                   tx_out <= 0;
                end
        if (tx_cnt > 0 && tx_cnt < 9) begin
                tx_out <= tx_reg[0];
                tx_reg[0]<=tx_reg[1];
                tx_reg[1]<=tx_reg[2];
                tx_reg[2]<=tx_reg[3];
                tx_reg[3]<=tx_reg[4];
                tx_reg[4]<=tx_reg[5];
                tx_reg[5]<=tx_reg[6];
                tx_reg[6]<=tx_reg[7];
                end
                if (tx_cnt == 9) begin
                   tx_out <= 1;
                   tx_cnt <= 0;
                   tx_empty <= 1;
                end
        end
        if ( ! tx_enable) begin
           tx_cnt <= 0;
        end
  end


endmodule
```

flexlogix Technologies, Inc.

Design Resource Usage and Performance

Design	LUT Utilization Percentage	LUTs	Flip Flops	IOs	Clock Domains	Clock Names	Max Frequency (MHz)			
							TSMC 40ULP tt1p1v25c_Typical	TSMC 28HPC+ tt0p9v25c_rctypical	TSMC 16FFC tt0p8v85c_rctypical	TSMC N7 tt0p75v25c_typical
APB 16550 UART	24.96%	629	243	48 inputs 43 outputs	1	pclk	76	306	430	493
APB simple UART	3.06%	77	63	15 inputs 10 outputs	2	PCLK rxclk	177 130	652 445	909 664	1303 878
SPI Master	2.62%	66	56	35 inputs 83 outputs	2	sclk pclk	136 205	516 956	760 1142	784 1360
32 bit GPIO APB	5.08%	128	130	43 inputs 64 outputs	1	PCLK	346	1644	2392	2433

Note: Performance is available for other process corners such as SS/TT/FF and -40 to 125C.