EFLX™ Customizable I/O on APB Bus

Updated March 7, 2023
Copyright 2023 Flex Logix Technologies, Inc.
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

SOC/MCU

- ARM Cortex M0
- FLASH
- SRAM
- ROM

System Bus

AHB-Lite Bus

AHB To APB Bridge

- PMU
- Counters
- EFLX
- IOs

EFLX as APB Slave

GPIO Buffers
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)
EFLX1K/4K I/O RBB

Inputs and outputs can be pass-through or flopped
EFLX with APB Slave Interface

23 min IOs / 78 max IOs
- 14 min/45 max Inputs
- 9 min/33 max Outputs

Notes:
1. Address width is application specific and can be less than 8
2. Data width can be 8-bits, 16-bits or 32 bits

0 when EFLX_PG = 1
(Need to isolate control signals when EFLX is powered off)
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:
APB Slave Interface Verilog Code

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

State Diagram

All I/Os to and from EFLX array are registered
APB 32-Bit GPIO Port

Fit into 1 EFLX1K Tile

APB Slave Interface

Address/Control

Decoder

RD/WR Control

32-bit Direction Register

RD/WR Data

32-bit Output Register

Interrupt Register

Interrupt Detection

32-bit Input Register

I/O Pads

I/O Buffers x32

APB BUS (PCLK, ...)

72 IOs

INTR

EFLX Array

flexlogix Technologies, Inc.
APB Simple UART

EFLX Array

APB Slave Interface

Control
Status
Register

Tx Data
Register

Rx Data
Register

Tx Shift
Register

Rx Shift
Register

RD/WR Data

Decoder

RD/WR Control

BAUD Rate
Generator

BAUD
Clock

1'b0

1'b1

24 IOs

APB BUS (PCLK, ...)

Fit into 1 EFLX1K Tile

I/O Pad

I/O Buffer

I/O Pad

I/O Buffer

flexlogix
Technologies, Inc.
APB 16550 UART

fit into 1 EFLX4K Tile

APB BUS (PCLK, …)

DDIS

1'b1

1'b0

INTR

25 IOs

Address

RD/WR

Din

Dout

Registers

RxBlock

Interrupt Control

Baudrate Generator

TxBlock

RxFIFO

TxFIFO

Modem Control

16550

16B Rx/Tx FIFO

Modem Control

4

6

SOUT

SOUT

I/O Pad

I/O Buffer

I/O Pad

I/O Buffer

I/O Pad
APB SPI Master

73 IOs

APB BUS (PCLK, ...)

Fit into 1 EFLX4K Tile
EFLX as an I/O Switch

Reconfigurable I/O Function

I^2C Controller

SPI Controller

Hard Wired I/Os

I/O Buffers

I/O Pads

APB BUS (PCLK, ...)

EFLX Array

Reconfigurable I/O Switch
APB 32-bit GPIO Example
APB Simple UART Example

```
// UART RX Logic
always # (positive edge PRESETn)
if (PRESETn==1)
begin
  // Check if last rx data was not unloaded,
  if (rx_empty) begin
    // Add rx data to end of buffer
    rx_data <= rx_data + rx_data;
    rx_empty <= 1;
    // Continue
    rx_data <= rx_data;
    rx_empty <= 1;
  end
else
begin
  // Receive data only when rx is enabled
  if (rx_enable)
  begin
    // Receive data
    rx_data <= rx_data + rx_data;
    rx_empty <= 1;
  end
end
```

```
// UART TX Logic
always # (positive edge PRESETn)
if (PRESETn==1)
begin
  tx_reg <= 0;
  tx_empty <= 1;
  tx_over_run <= 0;
  tx_out <= 1;
  baudcnt <= 1;
else
begin
  // Clock if just received start of frame
  if (rx_empty)
  begin
    tx_reg <= tx_reg + tx_reg;
    tx_empty <= 1;
    // Start of frame detected, proceed with rest of data
    if (tx_enable)
    begin
      // Logic to sample at middle of data
      if (tx_data)
      begin
        // Check if last rx data was not unloaded,
        if (rx_empty)
        begin
          // Add rx data to end of buffer
          rx_data <= rx_data + rx_data;
          rx_empty <= 1;
          // Continue
          rx_data <= rx_data;
          rx_empty <= 1;
        end
        else
        begin
          // Receive data only when rx is enabled
          if (rx_enable)
          begin
            // Receive data
            rx_data <= rx_data + rx_data;
            rx_empty <= 1;
          end
        end
      end
    end
  end
end
```

```
## Design Resource Usage and Performance

<table>
<thead>
<tr>
<th>Design</th>
<th>LUT Utilization Percentage</th>
<th>LUTs</th>
<th>Flip Flops</th>
<th>IOs</th>
<th>Clock Domains</th>
<th>Clock Names</th>
<th>Max Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TSMC 40ULP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>tt1p1v25c_Typical</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TSMC 28HPC+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>tt0p9v25c_rctypical</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TSMC 16FFC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>tt0p8v85c_rctypical</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TSMC N7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>tt0p75v25c_rctypical</td>
</tr>
<tr>
<td>APB 16550 UART</td>
<td>24.96%</td>
<td>629</td>
<td>48 inputs</td>
<td></td>
<td>1</td>
<td>pclk</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>243 43 outputs</td>
<td></td>
<td>1</td>
<td></td>
<td>306</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>430</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>493</td>
</tr>
<tr>
<td>APB simple UART</td>
<td>3.06%</td>
<td>77</td>
<td>15 inputs</td>
<td></td>
<td>2</td>
<td>PCLK</td>
<td>177</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>63 10 outputs</td>
<td></td>
<td>2</td>
<td></td>
<td>652</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>909</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1303</td>
</tr>
<tr>
<td>SPI Master</td>
<td>2.62%</td>
<td>66</td>
<td>35 inputs</td>
<td></td>
<td>2</td>
<td>sclk</td>
<td>136</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>56 83 outputs</td>
<td></td>
<td>2</td>
<td></td>
<td>516</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>760</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>784</td>
</tr>
<tr>
<td>32 bit GPIO APB</td>
<td>5.08%</td>
<td>128</td>
<td>43 inputs</td>
<td></td>
<td>1</td>
<td>PCLK</td>
<td>346</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>13064 outputs</td>
<td></td>
<td>1</td>
<td></td>
<td>1644</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2392</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2433</td>
</tr>
</tbody>
</table>

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