Reconfigurable Architecture (8)

osana@eee.u-ryukyu.ac.jp

Making the stopwatch

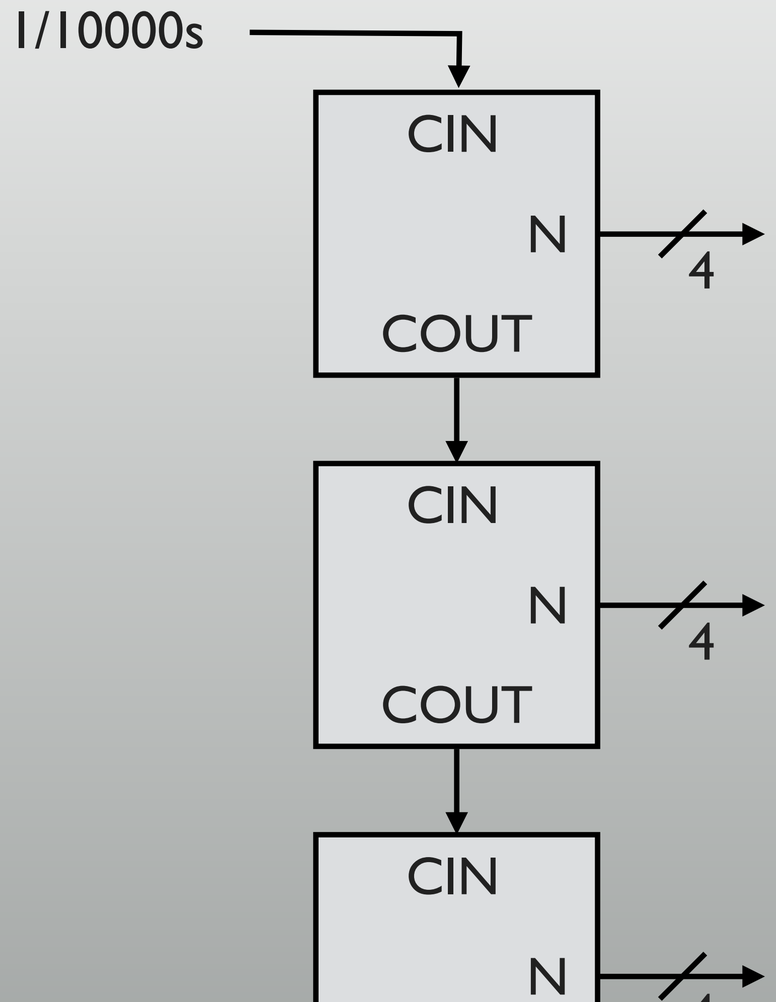
- 2 major approaches:
  - Preserving the original reg [31:0] CNT;
  - Replacing the register + counter with decimal counter module(s)

Preserving the 32bit reg CNT

- always @ (posedge CLK) CNT <= CNT+1;
  - Requires “divide by” or “mod of” 10 operations
    - ex: `assign val[27:24] = cnt2 / 10_000_00 % 10 ;`
  - Integer dividers are usually too large and slow
    - Each column requires separate divider…!

Making + cascading a decimal counter

```
module d_counter
  ( input RST, CLK, CIN,
    output wire COUT, reg [3:0] N);

   always @ (posedge CLK) begin

      if (RST) N <= 0;
      else
        if (CIN) N <= (N!=9) ? N+1 : 0;

   end
   assign COUT = (N==9) & CIN;
endmodule
```

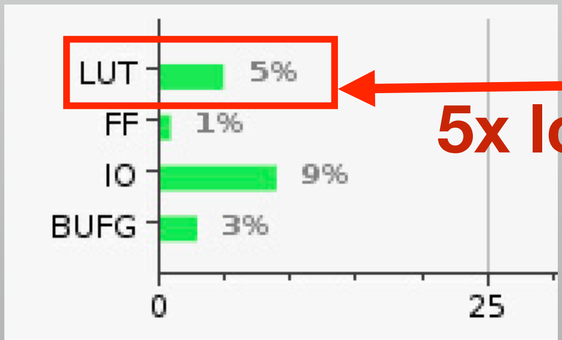


Comparing the results

32bit counter + dividers:

Timing	
Worst Negative Slack (WNS):	-15.102 ns
Total Negative Slack (TNS):	-312.319 ns
Number of Failing Endpoints:	31
Total Number of Endpoints:	628
Implemented Timing Report	

Timing is not met!

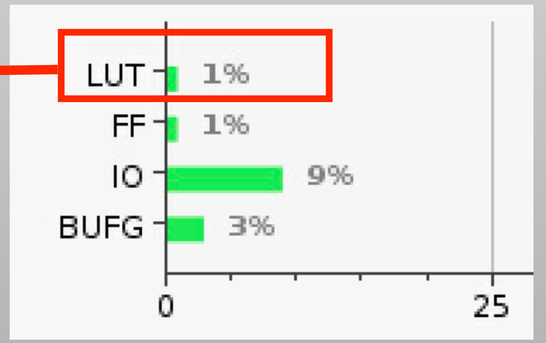


Design is straightforward,
but too slow & large

Decimal counter:

Timing	
Worst Negative Slack (WNS):	5.342 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	220
Implemented Timing Report	

5x logic size



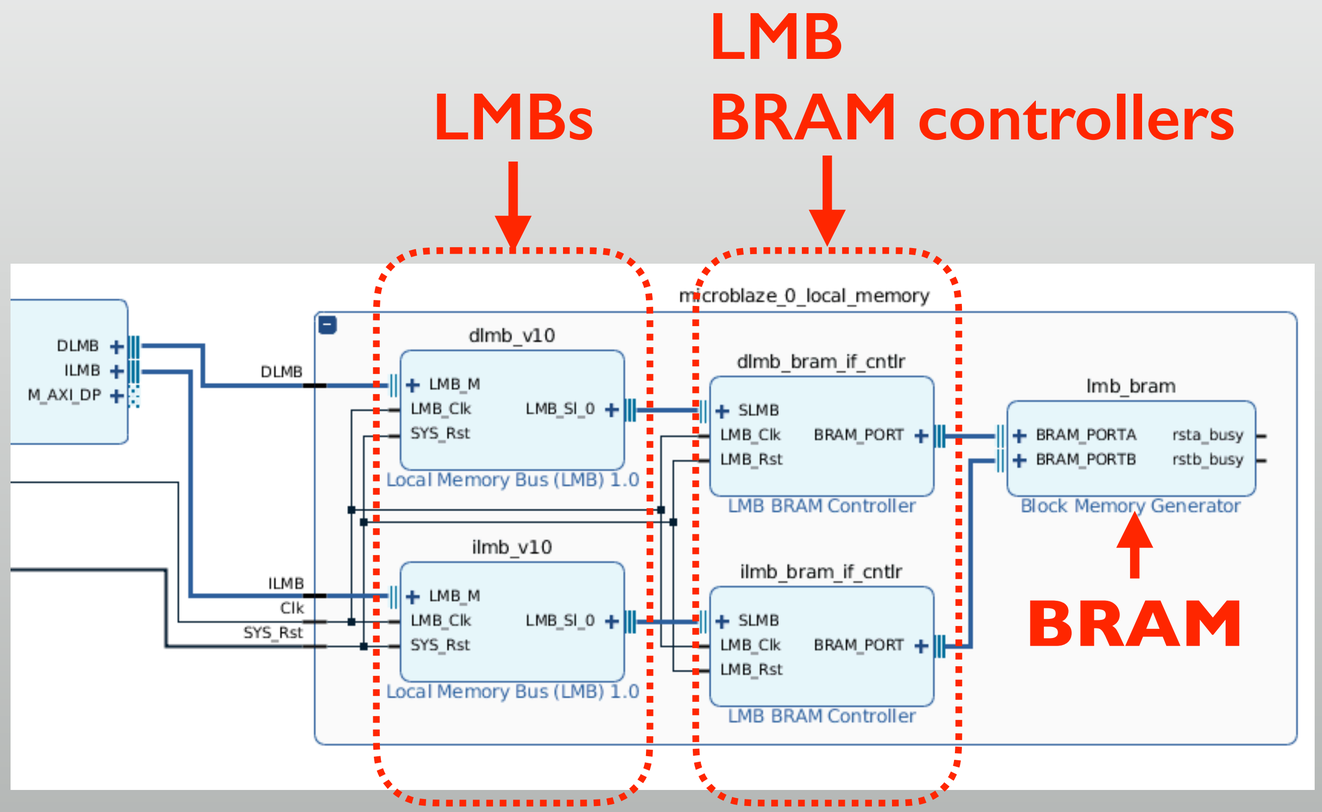
Requires re-design of the counter,
but small & fast

Last week and Today

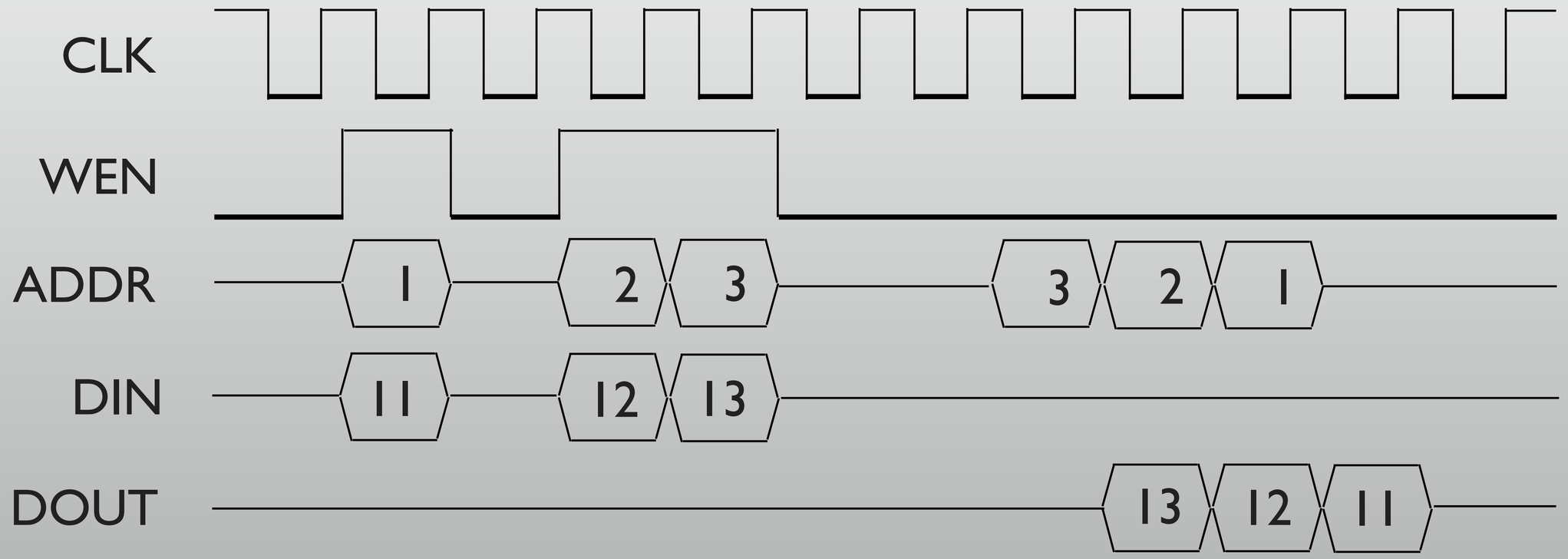
- Last week: minimum processor-based system
  - CPU + BlockRAM + AXI UART-Lite
  - “Hello World” from SDK
- Today: processor-based system + RTL design

MicroBlaze's default memory system

- (LMB + BRAM controller) x2
  - ILMB (Instruction LMB)
  - DLMB (Data LMB)
- Share the same dual-port BRAM
  - address range
    0x0000_0000 - 0x0000_ffff



BRAM R/W interface



BRAM-Like interface on RTL module

- Write-only interface
  - Holds a 32bit value
  - on address 0x0001_0000

```
module reg32w
  ( input wire CLK,
    input wire WE,
    input wire [31:0] A, DIN,
    output reg [31:0] VAL );

  always @ (posedge CLK) begin
    if (WE & A==32'h0001_0000)
      VAL <= DIN;
  end
endmodule
```

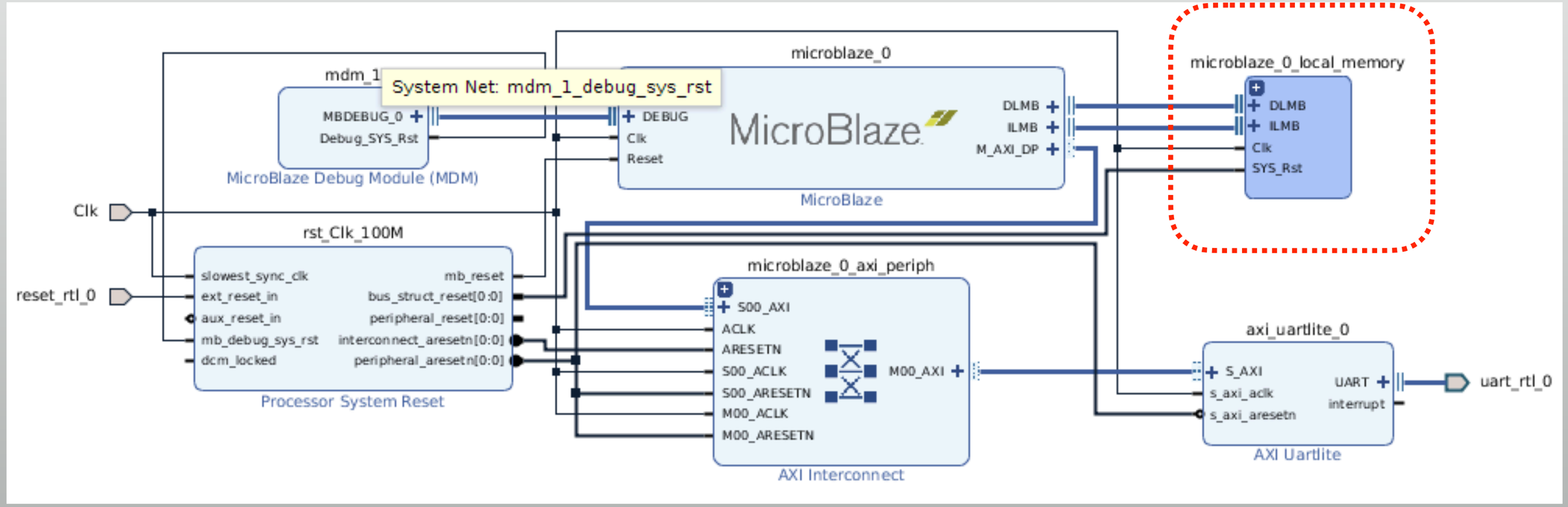
BRAM-Like R/W interface in RTL

- Separate read interface
  - 1 clock latency required
  - 1 of 2 values goes to Blaze
    - VAL1 @ 0x0001_0004
    - VAL2 @ 0x0001_0008

```
module reg32r
  ( input wire CLK,
    input wire [31:0] A,
    output reg [31:0] DOUT,
    input wire [31:0] VAL1, VAL2 );

  always @ (posedge CLK) begin
    DOUT <= (A==32'h0001_0004) ? VAL1 :
            (A==32'h0001_0008) ? VAL2 : 0;
  end
endmodule
```

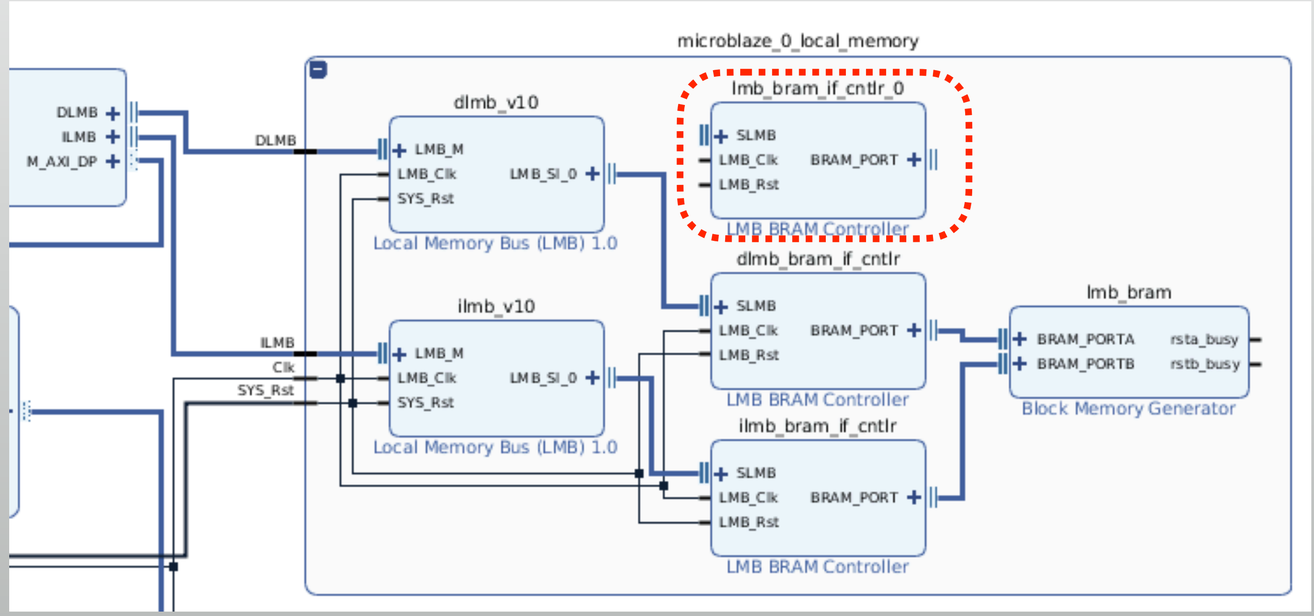
More BRAM interface on MBlaze

* Block diagram in last week



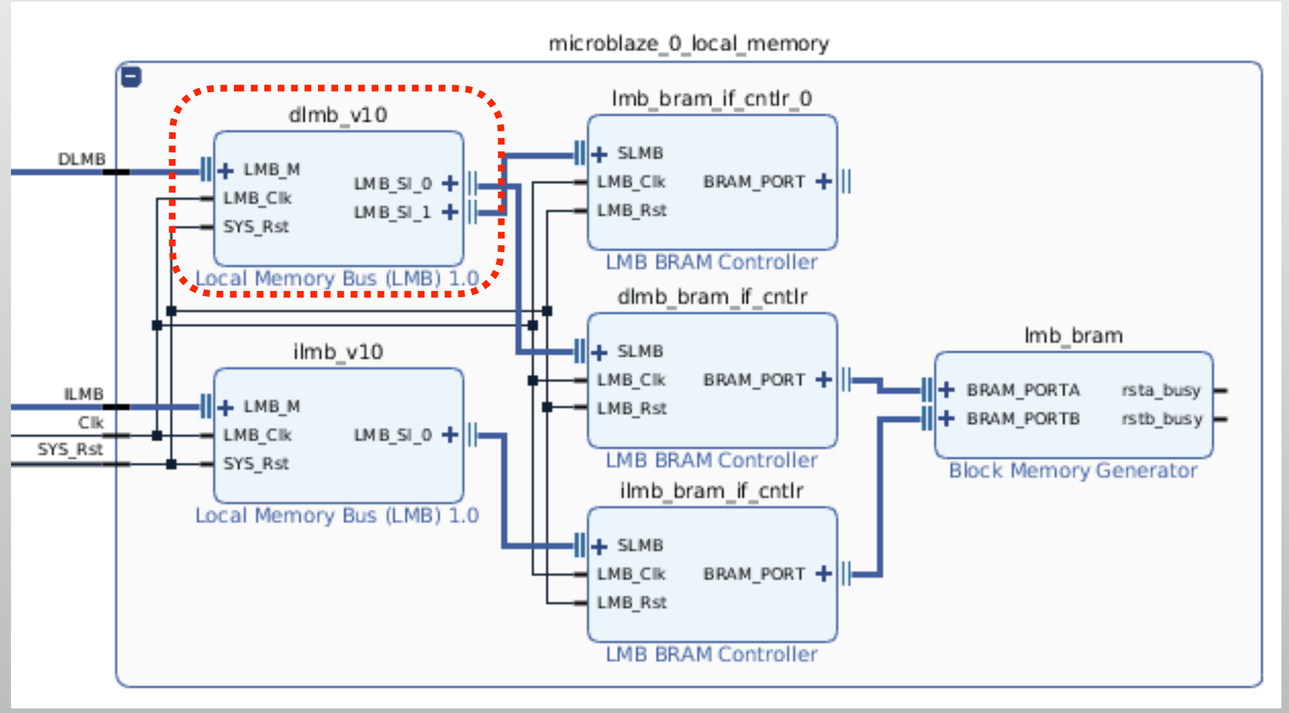
More BRAM controller: step 2

- Expand “Local_memory”
  - Add an “LMB BRAM Controller”
  - Move it in the local memory



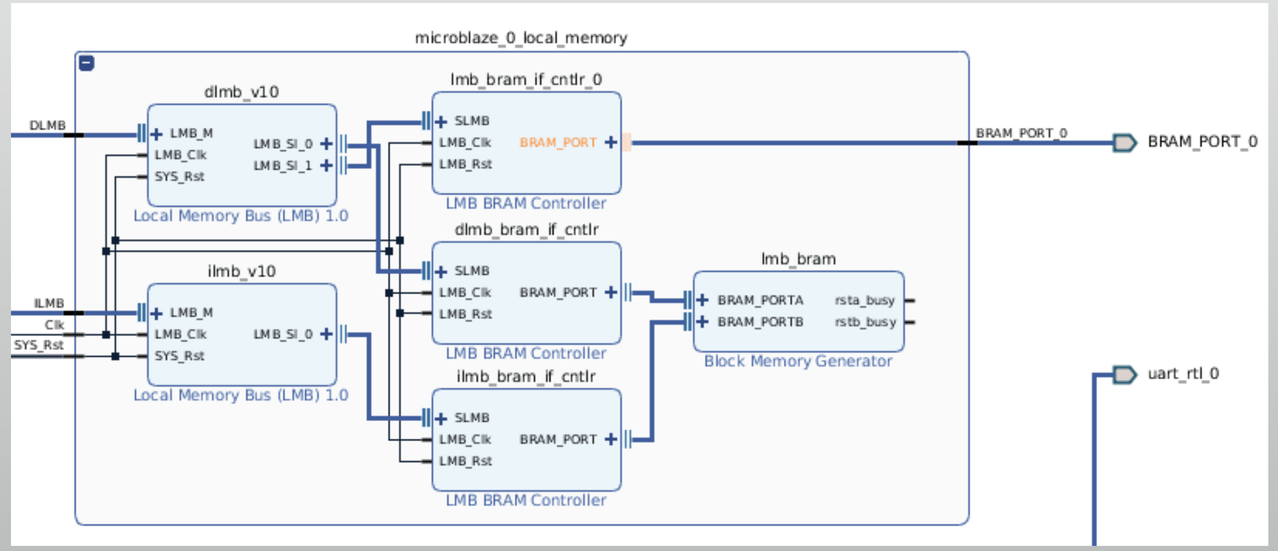
More BRAM controller: step 3

- Double-click on dlmb_v10
  - Change # slaves: 1 → 2
  - Then connect LMB_SI_1 to the new LMB BRAM controller
  - Also connect clk + reset signals



More BRAM controller: step 4

- Right-click on the “BRAM_PORT” of the new LMB BRAM controller
  - “Make external” and you’ve done
  - Address still not mapped



More BRAM controller: step 5

- Open the address map
  - New LMB BRAM in “Unmapped slaves”
  - Right-click → Assign
    - 8KB default

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
Unmapped Slaves (1)					
microblaze_0_local_memory/lmb_bram_if_cntlr_0	SLMB	Mem			
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
microblaze_0_local_memory/lmb_bram_if_cntlr_0	SLMB	Mem	0x0001_0000	8K	0x0001_1FFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF

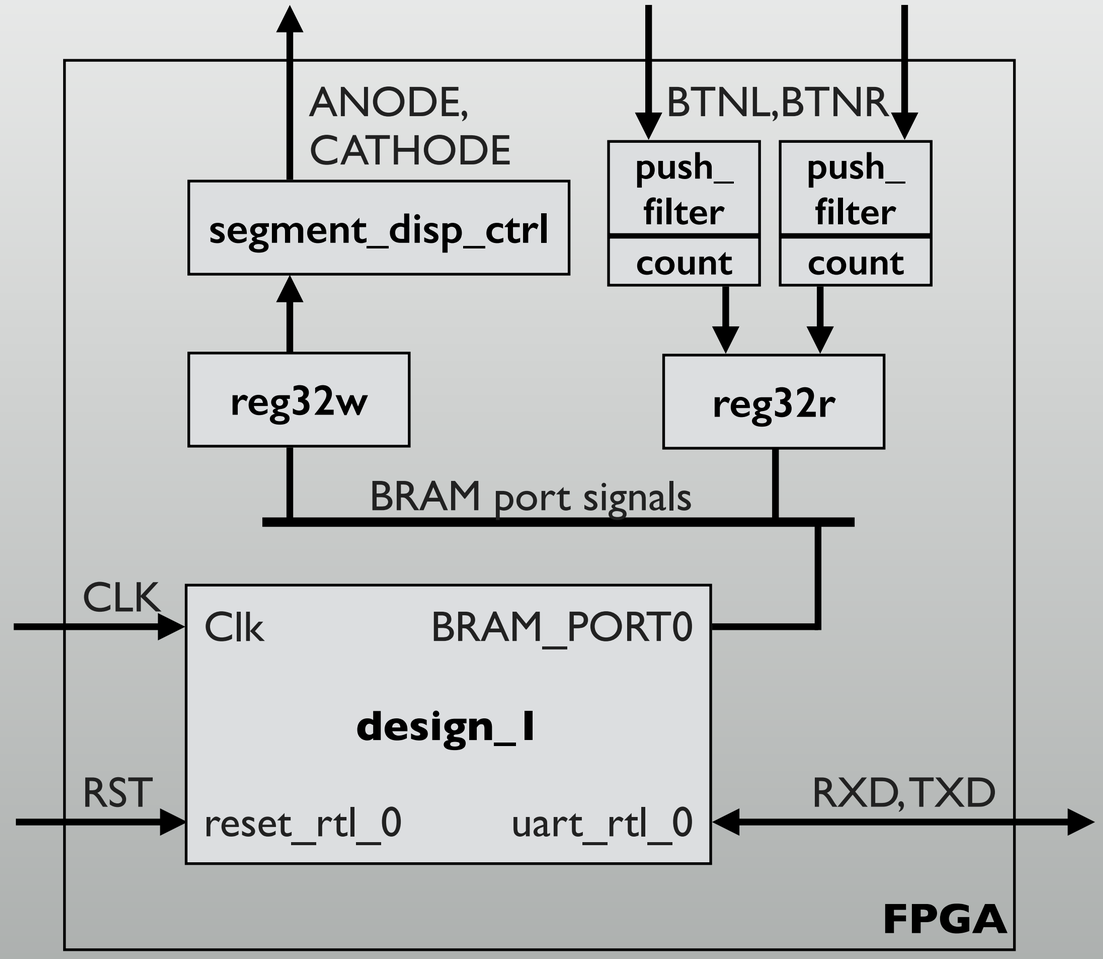
More BRAM controller: step 6

- Create HDL wrapper again
  - BRAM related ports appears on the wrapper module ports

```
24      output [0:31]BRAM_PORT_0_addr;
25      output BRAM_PORT_0_clk;
26      output [0:31]BRAM_PORT_0_din;
27      input [0:31]BRAM_PORT_0_dout;
28      output BRAM_PORT_0_en;
29      output BRAM_PORT_0_rst;
30      output [0:3]BRAM_PORT_0_we;
31      input Clk;
32      input reset_rtl_0;
33      input uart_rtl_0_rxd;
34      output uart_rtl_0_txd;
```

Organizing the system

- Connect reg32w and reg32r to BRAM_PORT0
  - reg32w to 7seg
  - 2 button counters to reg32r



Source files

- lab1/
  - 7seg.v: Everything need for 7 segment display
  - push_button.v: Everything of push button filter
  - reg32.v: reg32w + reg32r
  - top.v
  - top.xdc (changed from last week)

Program in SDK

- 7 segment display:
  - Show # key
- Serial:
  - Show button count on key press
- Replace main() of Hello world example

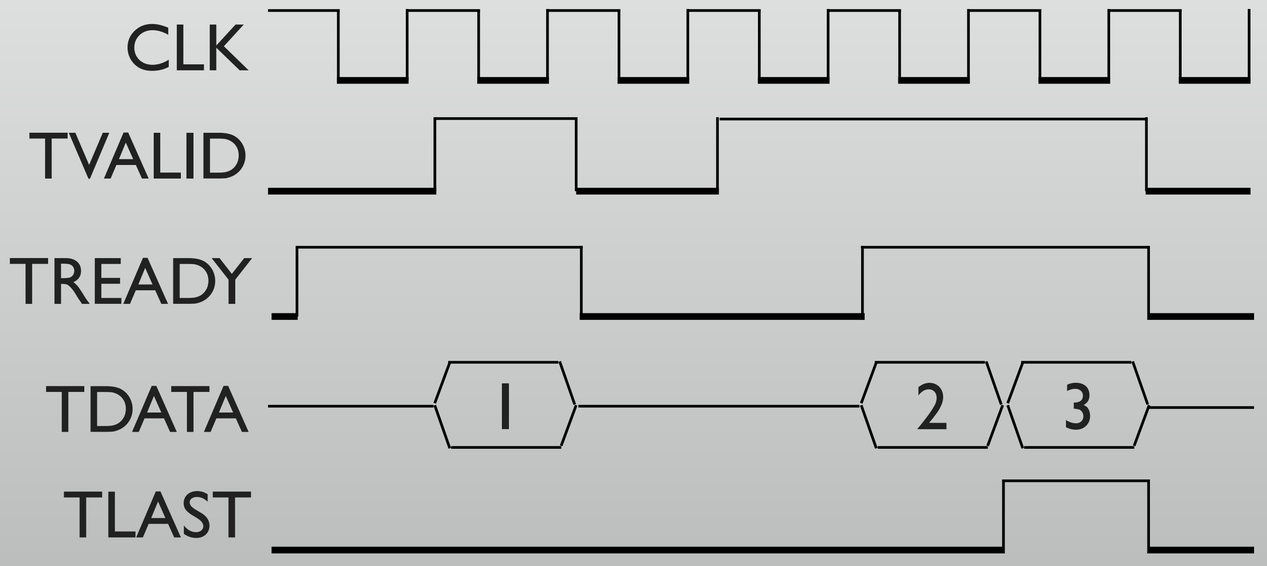
```
// no buffer for getchar()
setvbuf(stdin, NULL, _IONBF, 0);

int *LED  = (int*)0x00010000;
int *BTNL = (int*)0x00010004;
int *BTNR = (int*)0x00010008;

int a=0;
while(1){
  xil_printf("L: %d R: %d a: %d \n\r”,
              *BTNL, *BTNR, a);
  *LED = a++;
  getchar();
}
```

More to do: AXI Stream

- Stream type interface
  - Good for transferring large data between CPU and custom logic
  - MicroBlaze and many Xilinx IP cores support AXI Stream
    - Data transfer APIs offered with Xilinx SDK
- Memory type interface is simple, but becomes complicated for large transfer

Handshake on AXI Stream

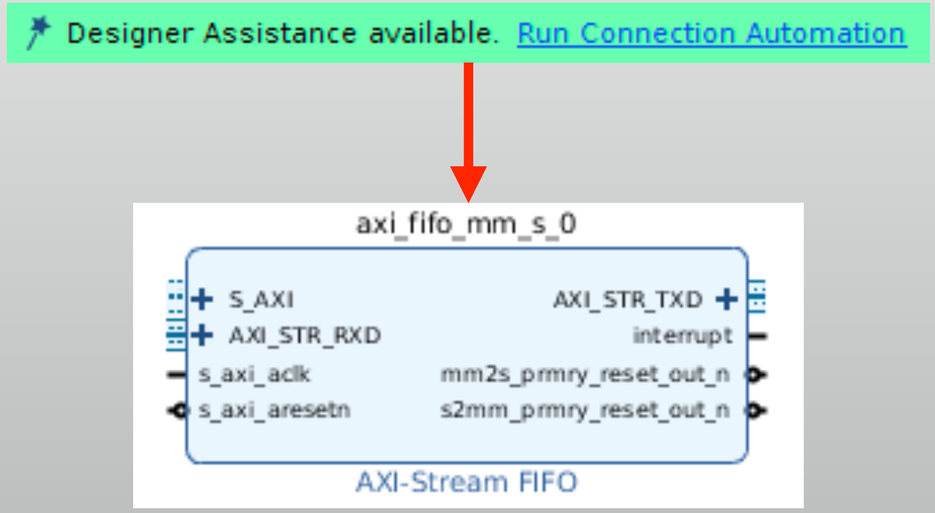
- Transfer on TVALID & TREADY
  - TDATA: Data (src→dest)
  - TVALID: Data valid (src→dest)
  - TREADY: Dest ready (dest→src)
  - TLAST: Last word of stream (src→dest)



Example: 3 word transfer

Add AXI Stream interface on MBlaze

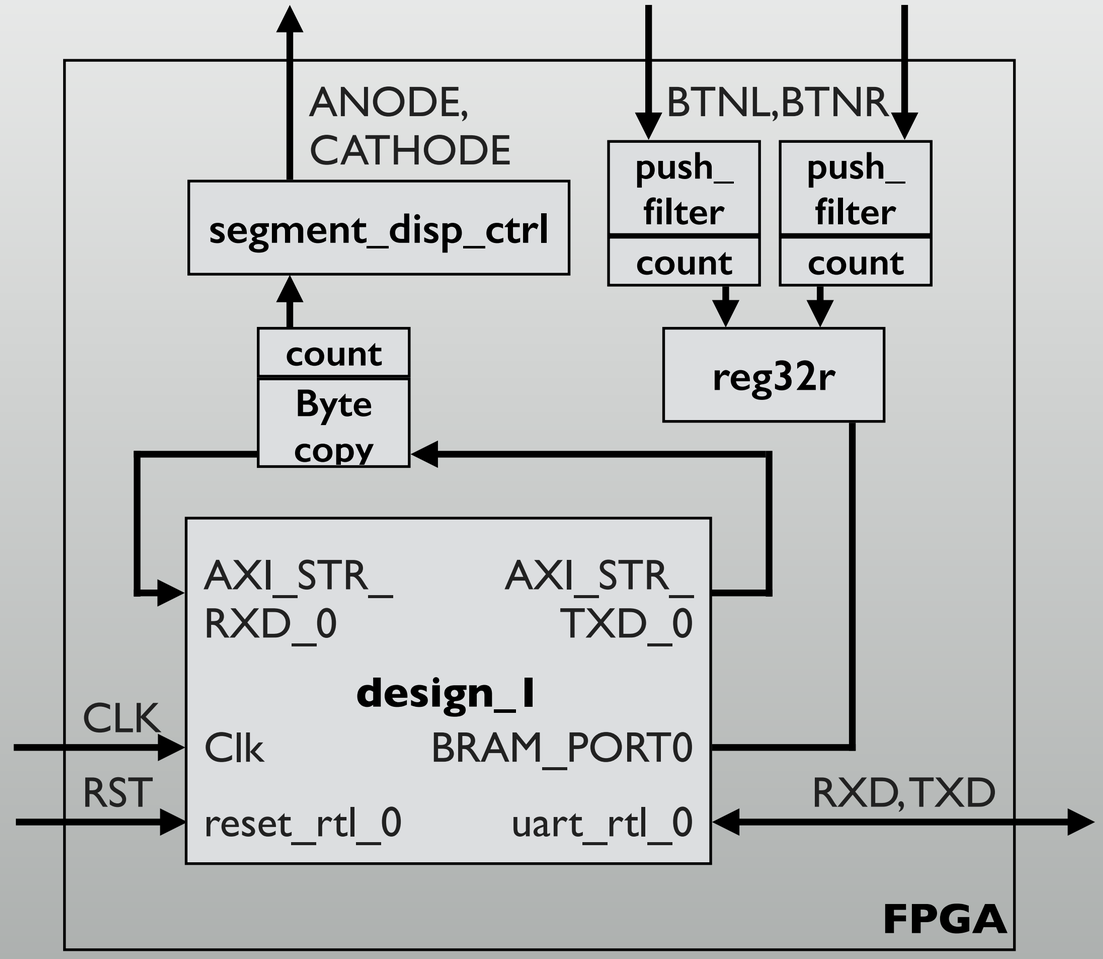
- Add “AXI-Stream FIFO”
  - Turn “Enable Transmit Control” off
  - Run connection automation to connect with MBlaze
  - Make external on AXI_STR_RXD and AXI_STR_TXD
- Create HDL wrapper again



Lab 2

Organizing the system

- AXI TXD → AXI RXD loopback
- Data count
  - if (TREADY & TVALID) CNT++;
- Data byte copy
  - AXI_RXD =
    { 4{AXI_TXD[7:0] };



Source files

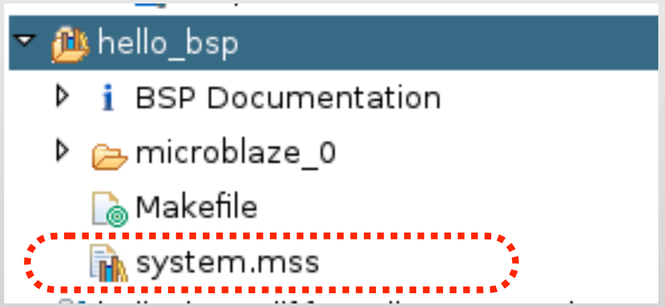
- lab2/
  - top-fifo.v: Top level module with AXI Stream loopback
  - cv(&FifoInstance, DestinationBuffer);: Sample AXI transfer code

fifo-example.c

- init() for API initialization
- send() to send data (with length to send)
- recv() to receive (API detects the length of received data frame)
  - Just modify main() for your own AXI stream logic

Access to BSP Documents / sample

- Find BSP in SDK's Project explorer
  - Open "System.mss" for BSP setting summary
  - "Peripheral Drivers" section has Documents and Examples



Peripheral Drivers

Drivers present in the Board Support Package.

axi_fifo_mm_s_0	llfifo	Documentation	Import Examples
axi_uartlite_0	uartlite	Documentation	Import Examples
microblaze_0_local_memory_dlmb_bram_if_cntlr	bram	Documentation	Import Examples
microblaze_0_local_memory_ilmb_bram_if_cntlr	bram	Documentation	Import Examples
microblaze_0_local_memory_lmb_bram_if_cntlr_0	bram	Documentation	Import Examples

Design recommendations

- For small / low-latency access between CPU-FPGA:
  - “Memory mapped” I/Os on BRAM port
  - No complicated device driver or APIs, just access through C pointers
- For large data transfer between CPU-FPGA:
  - Use DMA (=AXI Stream Data FIFO)
  - Access through API, simple example in fifo-example.c in today’s lab2.