HW#3 Adding an I/O Controller

Chun-Jen Tsai
National Chiao Tung University
11/17/2020

Homework Goal

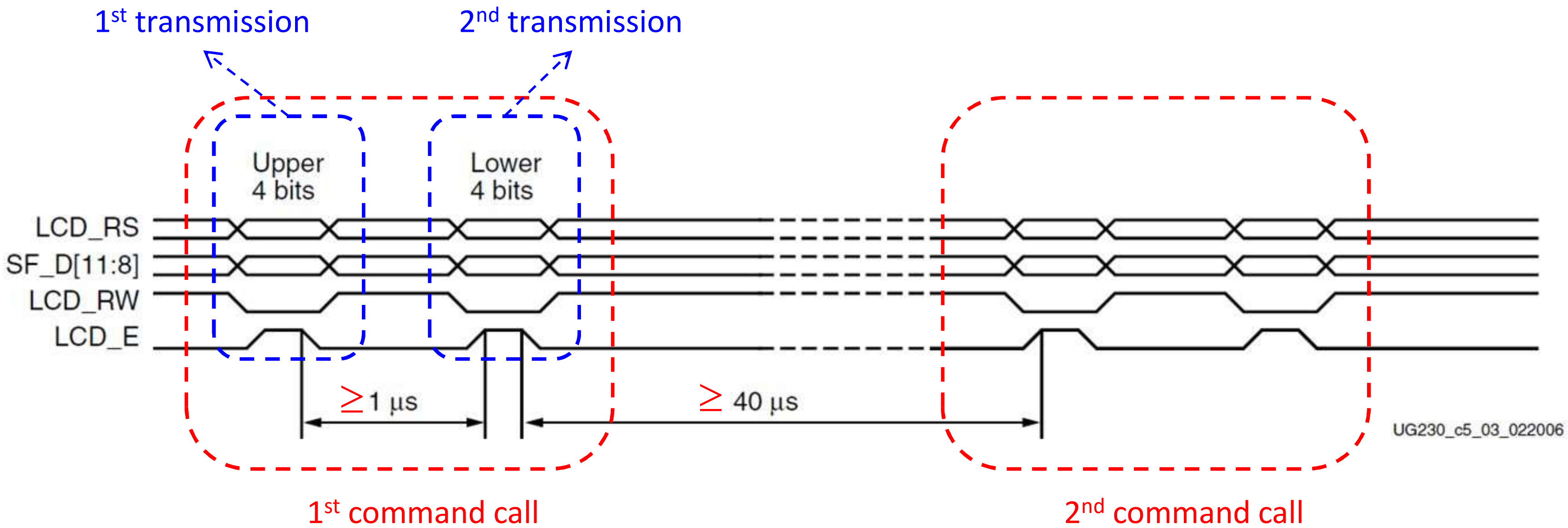
- Add a text-LCD controller to the Aquila SoC, and provide a demo program that print to the LCD screen
- Two ways to design the controller:
  - A GPIO device (simple HW, complex SW)
  - A text-screen buffer device (complex HW, simple SW)
- Upload your report and HW/SW code to E3 by 11/30, 17:00.

Review on 1602 LCD Device

- There is an 1602 LCD display on the Aquila platform
  - Displays two-lines of text, each has 16 characters.
  - Operating in 4-bit 1602 LCD mode
  - The LCD is driven by 3 control wires (LCD_E, LCD_RS, LCD_RW) and 4 data wires (DB4~DB7)

Character LCD Interface

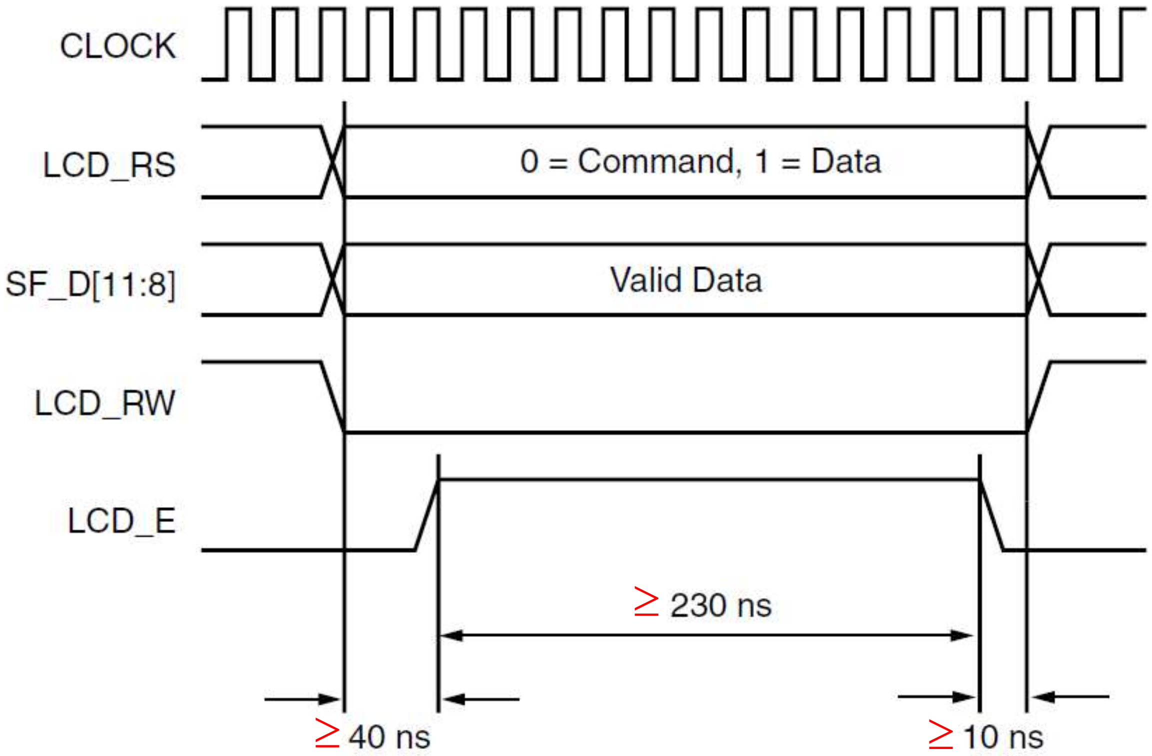
- For the 4-bit operating mode:
  - Each command will need two transmissions, using only E, RS, RW, and DB4~DB7
  - For example, to write a ASCII to the current cursor position, you must set RS to 1, RW to 0, then sent 8-bit ASCII code in two transmissions



Ref: Figure 5-6, *Spartan-3E FPGA Starter Kit Board User Guide*, Xilinx UG230, Jan. 2011.

Timing Diagrams for Transmission

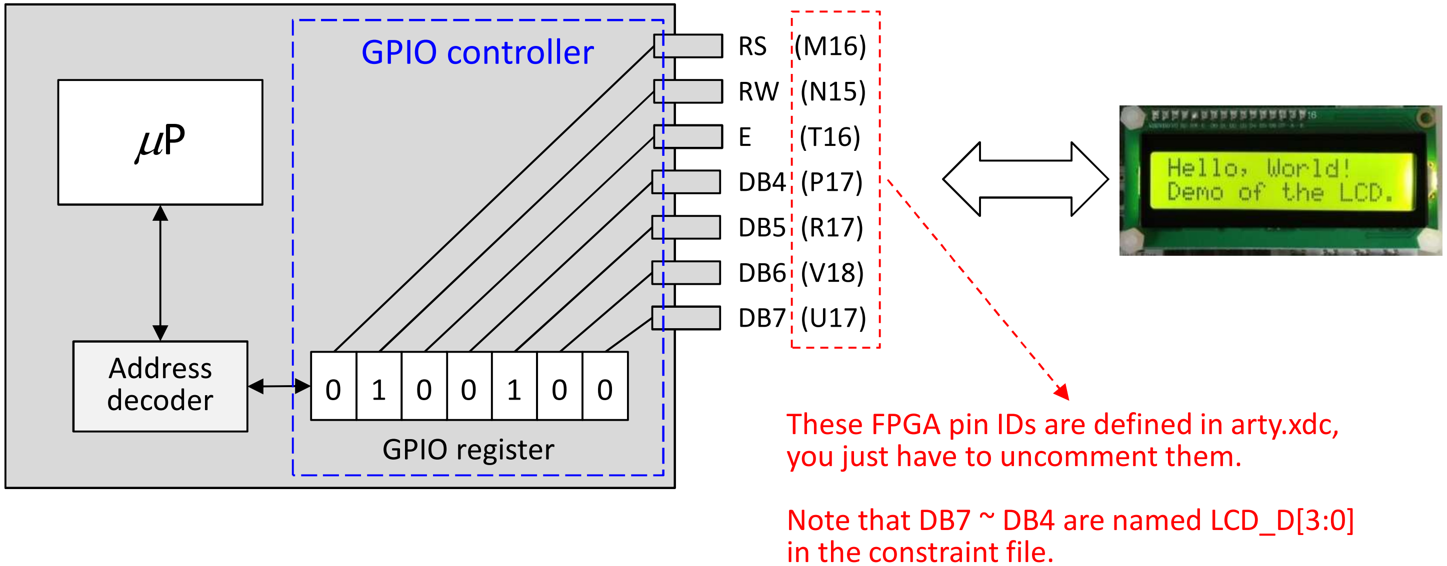
- The timing of one transmission in 4-bit mode:
  - Note that execution of a function requires two transmissions



Ref: Figure 5-6, *Spartan-3E FPGA Starter Kit Board User Guide*, Xilinx UG230, Jan. 2011.

Method #1: GPIO Controller

- A GPIO device allows a microprocessor to generate specific waveforms
  - The switching frequency is limited by the CPU clock rate
  - Fast enough for the 1602 LCD module
  - Typical GPIO ports (RS ~ DB7) are declared as `inout` ports. However, `output` declaration is good enough for us here.



μP Interface for GPIO Register

- A design example would be the CLINT device

```
module clint
#( parameter TIMER = 100_000, parameter XLEN = 32 )
(
   input                          clk_i,
   input                          rst_i,

   input                          en_i,
   input                          we_i,
   input [2 : 0]                  addr_i,
   input [XLEN-1 : 0]             data_i,
   output reg [XLEN-1 : 0]        data_o,
   output reg                     data_ready_o,

   output                         tmr_irq_o,
   output                         sft_irq_o
);

reg  [XLEN-1 : 0] clint_mem[0: 4];
wire [63: 0] mtime = { clint_mem[1], clint_mem[0] };
wire [63: 0] mtimecmp = { clint_mem[3], clint_mem[2] };
wire [XLEN-1 : 0] msip = clint_mem[4];
```

```
always @(posedge clk_i)  /* register write */
begin
   if (rst_i)
   begin
      clint_mem[0] <= 32'b0; clint_mem[1] <= 32'b0;
      clint_mem[2] <= 32'b0; clint_mem[3] <= 32'b0;
      clint_mem[4] <= 32'b0;
   end
   else if (we_i)
      clint_mem[addr_i] <= data_i;
         . . .
end

always @(posedge clk_i)  /* register read */
begin
   if (en_i)
   begin
      data_o <= clint_mem[addr_i];
      data_ready_o <= 1;
   end
   else
      data_ready_o <= 0;
end
```

Address Assignment

- We can assign an address to the register in the bus decoder logic (in `aquila_top.v`)

```
// ----------- System Memory Map: DDRx DRAM, Devices, or CLINT --------------
//      [0] 0x0000_0000 - 0x0FFF_FFFF : Tightly-Coupled Memory (TCM)
//      [1] 0x8000_0000 - 0xBFFF_FFFF : DDRx DRAM memory (cached)
//      [2] 0xC000_0000 - 0xCFFF_FFFF : device memory (uncached)
//      [3] 0xF000_0000 - 0xF000_0010 : CLINT I/O registers (uncached)
//
wire [3 : 0] code_segment, data_segment;

assign data_segment = p_d_addr[XLEN-1:XLEN-4];

assign data_sel = (data_segment == 4'h0)? 0 :
                  (data_segment == 4'hC)? 2 :      // used by UART
                  (data_segment == 4'hF)? 3 :      // used by CLINT
                                          1;
```

```
clint #( .TIMER(50_000) )
CLINT(
   .clk_i(clk_i),
   .rst_i(rst_i),
   .en_i(data_sel == 3),
   .we_i(data_rw && (data_sel == 3)),
   .addr_i({6'b0, p_d_addr[XLEN - 5 : 2]}),
   .data_i(p_d_core2mem),
   .data_o(data_from_clint),
   .data_ready_o(clint_d_ready),

   .tmr_irq_o(tmr_irq),
   .sft_irq_o(sft_irq)
);
```

LCD Control in SW

- For 1602 LCD I/O software using GPIO device, you can Google for the example of Arduino
  - For example, to send an ASCII code to the LCD:

```
void digitalWrite(int port, Xuint32 value)
{
   Xuint32 temp = gpio_read();
   temp &= (0xFFFFFFFF - port_mask[port]);
   temp |= value << port;
   gpio_write(temp);
}

void cputch(uint8_t code)
{
  digitalWrite(LCD_RS, 1);
  digitalWrite(LCD_RW, 0);

  write4bits(code>>4); // write upper nibble
  write4bits(code);     // write lower nibble
}
```

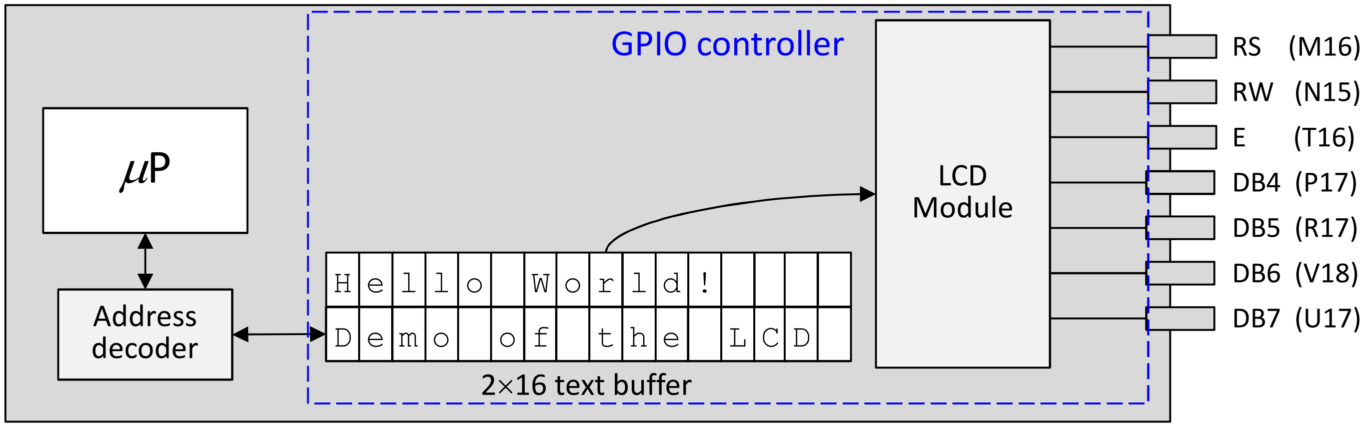
```
void write4bits(uint8_t value)
{
  digitalWrite(LCD_D4, (value >> 0) & 0x01);
  digitalWrite(LCD_D5, (value >> 1) & 0x01);
  digitalWrite(LCD_D6, (value >> 2) & 0x01);
  digitalWrite(LCD_D7, (value >> 3) & 0x01);

  // pulse enable
  digitalWrite(LCD_E, LOW);
  delayMicroseconds(1);
  digitalWrite(LCD_E, HIGH);
  delayMicroseconds(1);     // enable pulse must be >450ns
  digitalWrite(LCD_E, LOW);
  delayMicroseconds(100);  // commands need > 37us to settle
}
```

The delay function can be implemented using the clock() function.

Method #2: Text Screen Buffer Device

- Another way to implement the LCD controller is to design a frame buffer device
  - The control signals are generated by the HW controller
  - A text buffer array can be accessed by the CPU (like a memory device) to provide the content of the screen
  - The HW controller scans the text buffer a few times a second to update the LCD screen



The LCD Module

- The LCD module is a bit tricky to implement, but you can reuse the one you got in D-Lab (available on E3):

```
module LCD_module(
    input clk,
    input reset,
    input [127:0] row_A,
    input [127:0] row_B,
    output reg LCD_E,
    output reg LCD_RS,
    output reg LCD_RW,
    output reg [3:0]LCD_D);
    );
```

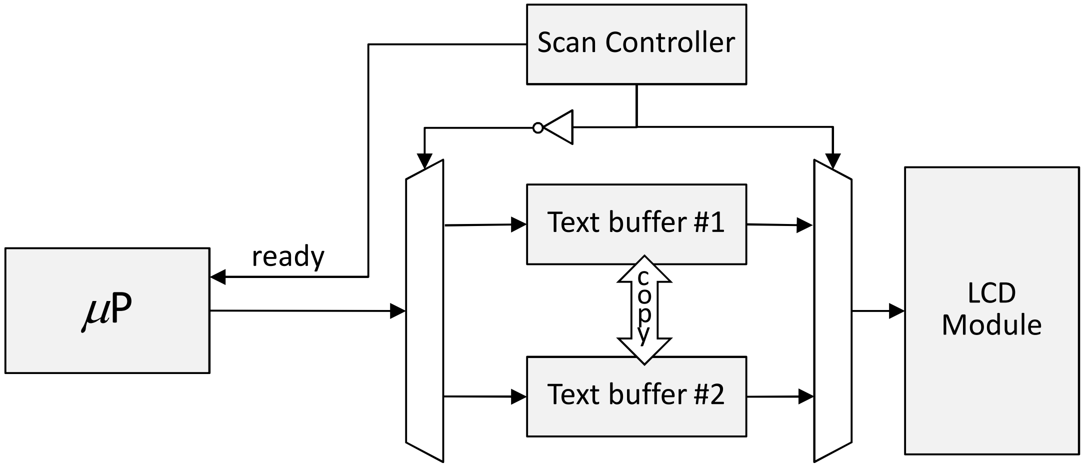
top-row of text

bottom-row of text

- What you need to do for Method #2:
  - Create a text buffer, wire it to `row_A` & `row_B`
  - Create an interface for the μP to write the text buffer

LCD Flickering Issue (Optional)

- Note that the LCD device is a slow device that can only be updated 5 ~ 6 times a second
- If μP write to the text buffer while the LCD is scanning & updating the screen, flicker may happen
  - You can use a pair of text buffers to resolve this issue
  - When the scan controller swap the buffer, it has to copy the buffers and pause the μP's write operation



Your Homework

- Implement the LCD controller using method #1, #2, or both. Also, provide a demo C code that print to the LCD
- Write a report discuss your implementation and problems you have encountered

- You must upload the `*.v` files of your controller and the `*.c`/`*.h` files of your demo program
  - I will use these files to make sure you do the homework by yourself.
  - Do not upload the workspace, just put new files and your modifications (e.g. `aquila_top.v`, `soc_top.v`) in a zip file.