ELEC 305

Digital System Design Lab

Fall 2024

Lecture 7:
Interfaces

Outline

- Interfaces: More than buttons, switches and LEDs (or not?)
- Analog: ADC / DAC
- “Digital” communication protocols: I2C, I2S, UART, SPI, …
- Graphics: VGA, TFT LCDs, …
- Basys3 GPIO project

Interfaces

- We’ve written interface requirements earlier, so what are the interfaces that we know of?
  - Buttons: digital input, monostable, we have to keep clicking to make it “stick”
  - Switches: digital input, bistable, we snap it once and it “sticks”
  - LEDs (incl. 7-seg displays): digital output, voltage controlled

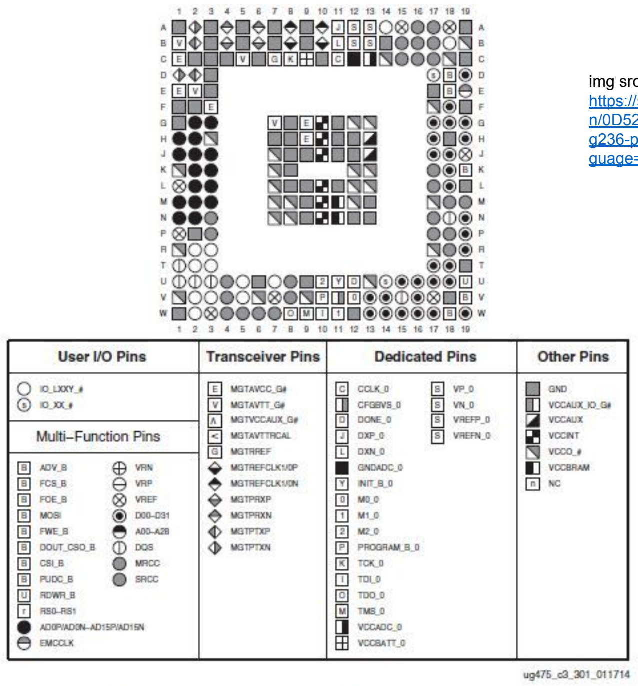
that’s about it

- These simple devices were tied to General Purpose Input Outputs (GPIO) on the FPGA

Interfaces

- On MCUs we know that GPIO is only one of many digital interfaces
  - e.g., Arduino Uno: GPIO, ADC, PWM, UART, I2C and SPI
- On the FPGA, other than a few special cases on more expensive FPGAs, there are no interfaces, all IO pins are GPIO.
- The MCUs actually have small circuits for those interfaces. On the FPGA we build these ourselves

CP236 and CPG236 Packages—XC7A15T, XC7A35T, and XC7A50T

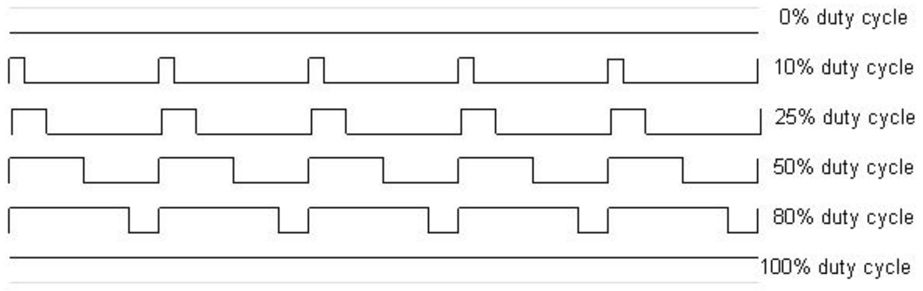


img src: https://support.xilinx.com/s/question/0D52E00006hphSgSAI/artix-7-cpg236-package-number-of-pins?language=en_US

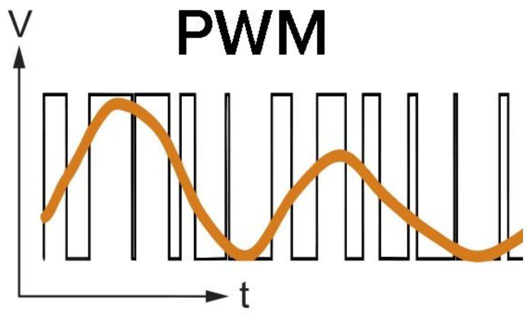
Figure 3-41: **CP236 and CPG236 Packages—XC7A15T, XC7A35T, and XC7A50T Pinout Diagram**

Interfaces

- A simple example: Pulse Width Modulation (PWM)
- Typically a fixed frequency "square wave", but the duty cycle (i.e., pulse width) is varied →→
- PWM is practically everywhere:
  - dimming lights, controlling motors, …
  - communications (e.g., duty cycle = message content)
  - …



img src: https://docs.arduino.cc/tutorials/generic/secrets-of-arduino-pwm/

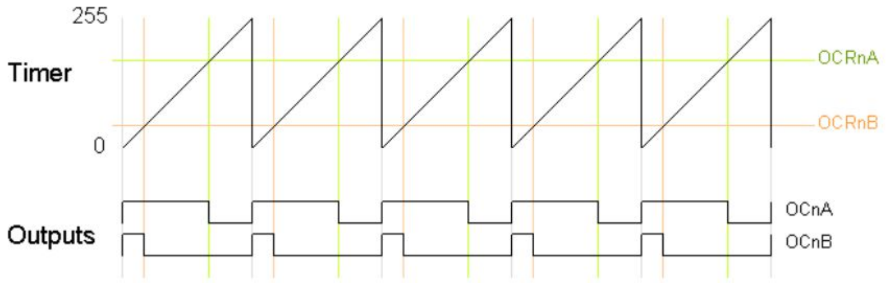


img src: https://www.thomsonlinear.com/tr/destek/ipuclari/pwm-nedir

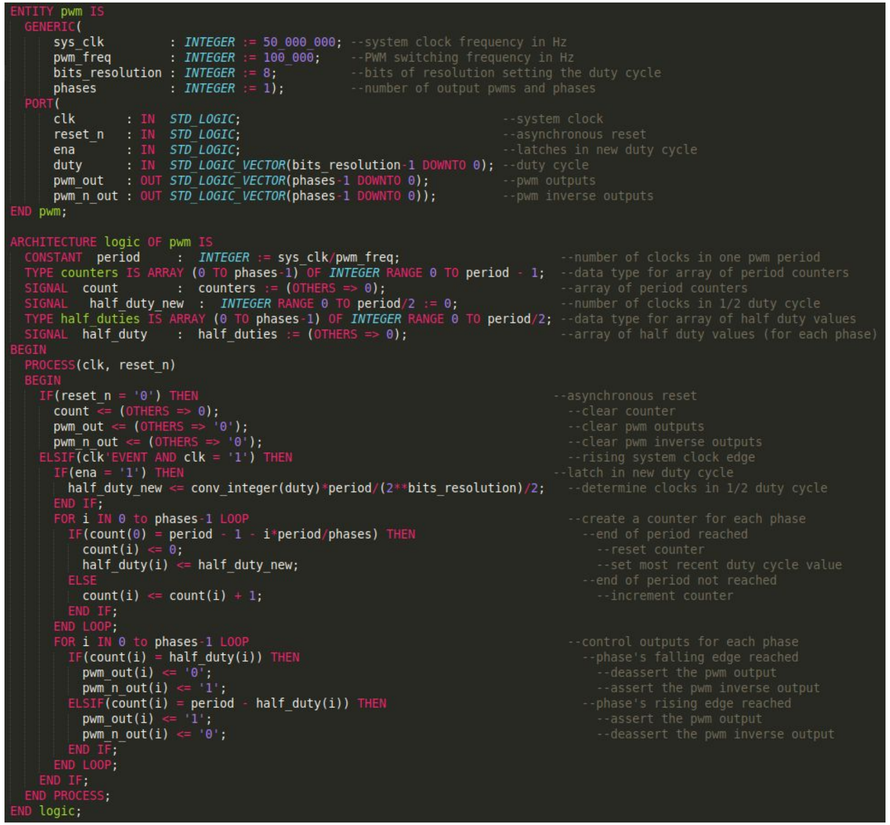
Interfaces

- On the MCU we generate the PWM waveform by generating an interrupt at timer values that correspond to pulse up and pulse down events

- On the FPGA it’s the same principle but no interrupts of course, just the circuit



img src: https://docs.arduino.cc/tutorials/generic/secrets-of-arduino-pwm/



```vhdl
ENTITY pwm IS
  GENERIC(
      sys_clk         : INTEGER := 50_000_000; --system clock frequency in Hz
      pwm_freq        : INTEGER := 100_000;    --PWM switching frequency in Hz
      bits_resolution : INTEGER := 8;          --bits of resolution setting the duty cycle
      phases          : INTEGER := 1);         --number of output pwms and phases
  PORT(
      clk       : IN  STD_LOGIC;                                    --system clock
      reset_n   : IN  STD_LOGIC;                                    --asynchronous reset
      ena       : IN  STD_LOGIC;                                    --latches in new duty cycle
      duty      : IN  STD_LOGIC_VECTOR(bits_resolution-1 DOWNTO 0); --duty cycle
      pwm_out   : OUT STD_LOGIC_VECTOR(phases-1 DOWNTO 0);          --pwm outputs
      pwm_n_out : OUT STD_LOGIC_VECTOR(phases-1 DOWNTO 0));         --pwm inverse outputs
END pwm;

ARCHITECTURE logic OF pwm IS
  CONSTANT  period     :  INTEGER := sys_clk/pwm_freq;                      --number of clocks in one pwm period
  TYPE counters IS ARRAY (0 TO phases-1) OF INTEGER RANGE 0 TO period - 1;  --data type for array of period counters
  SIGNAL  count        :  counters := (OTHERS => 0);                        --array of period counters
  SIGNAL   half_duty_new  :  INTEGER RANGE 0 TO period/2 := 0;              --number of clocks in 1/2 duty cycle
  TYPE half_duties IS ARRAY (0 TO phases-1) OF INTEGER RANGE 0 TO period/2; --data type for array of half duty values
  SIGNAL  half_duty    :  half_duties := (OTHERS => 0);                     --array of half duty values (for each phase)
BEGIN
  PROCESS(clk, reset_n)
  BEGIN
    IF(reset_n = '0') THEN                                                 --asynchronous reset
      count <= (OTHERS => 0);                                                --clear counter
      pwm_out <= (OTHERS => '0');                                            --clear pwm outputs
      pwm_n_out <= (OTHERS => '0');                                          --clear pwm inverse outputs
    ELSIF(clk'EVENT AND clk = '1') THEN                                      --rising system clock edge
      IF(ena = '1') THEN                                                   --latch in new duty cycle
        half_duty_new <= conv_integer(duty)*period/(2**bits_resolution)/2;   --determine clocks in 1/2 duty cycle
      END IF;
      FOR i IN 0 to phases-1 LOOP                                            --create a counter for each phase
        IF(count(0) = period - 1 - i*period/phases) THEN                       --end of period reached
          count(i) <= 0;                                                         --reset counter
          half_duty(i) <= half_duty_new;                                         --set most recent duty cycle value
        ELSE                                                                   --end of period not reached
          count(i) <= count(i) + 1;                                              --increment counter
        END IF;
      END LOOP;
      FOR i IN 0 to phases-1 LOOP                                            --control outputs for each phase
        IF(count(i) = half_duty(i)) THEN                                       --phase's falling edge reached
          pwm_out(i) <= '0';                                                     --deassert the pwm output
          pwm_n_out(i) <= '1';                                                   --assert the pwm inverse output
        ELSIF(count(i) = period - half_duty(i)) THEN                           --phase's rising edge reached
          pwm_out(i) <= '1';                                                     --assert the pwm output
          pwm_n_out(i) <= '0';                                                   --deassert the pwm inverse output
        END IF;
      END LOOP;
    END IF;
  END PROCESS;
END logic;
```

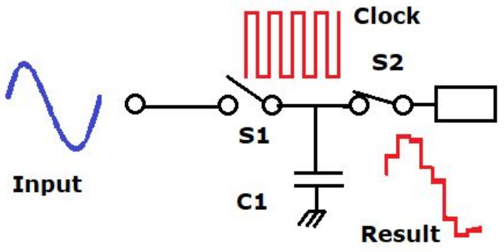
Interfaces

- Other interfaces are the same! If it’s digital, we can build a circuit for it on the FPGA
  - As long as the performance criteria allow it of course (sometimes the clock freq at which you have to work with is larger than what your device allows. In that case you’re out of luck)
- Also communication interfaces like I2C, I2S, UART, RX232, … all of these are just GPIOs that are actuated and read according a specific “protocol”
- In summary, we studied GPIOs so far (buttons, LEDs, switches), **but other digital interfaces are actually only more contrived versions of the same thing**
- I mentioned at the start of the semester → Most interfaces you will run into in this course will already have optimized VHDL modules available online. We’ll therefore study the protocols and just use available modules (maybe modify them a bit) and cite those.

Interfaces: Analog

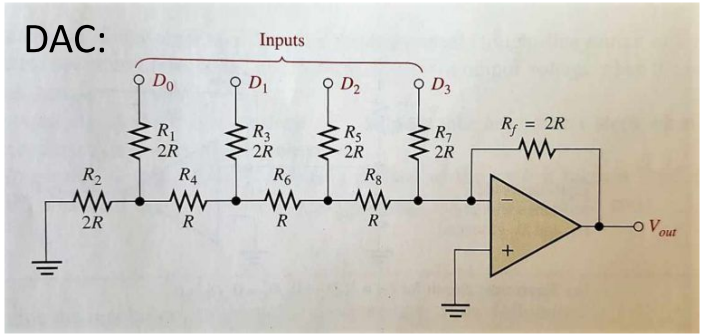
- We said “if it’s digital, we can build a circuit for it on the FPGA”, what about analog?
- Interfacing with analog signals requires similar actions, but additional circuitry is needed
- Specifically, an ADC needs at least a sample-and-hold circuit, and a DAC needs something like a resistive ladder
  - Designing these circuits for fast operation requires serious expertise
- Processing the sampled versions of the signals is the same as the other digital interfaces we’ve seen, just “bits moving up and down”

ADC:



img src: https://www.circuitbasics.com/analog-to-digital-converters/

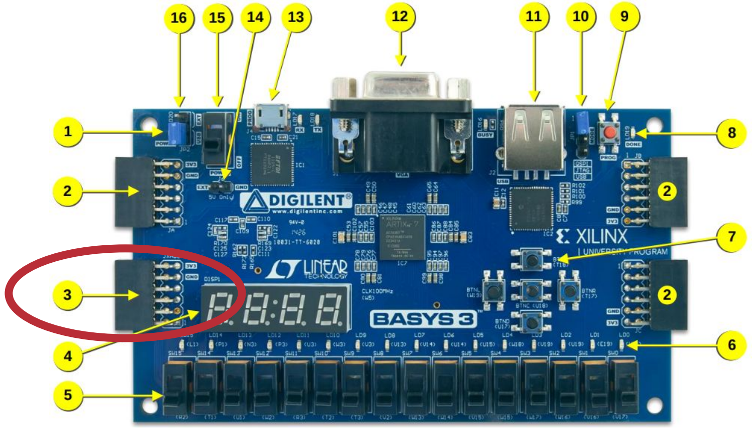
DAC:



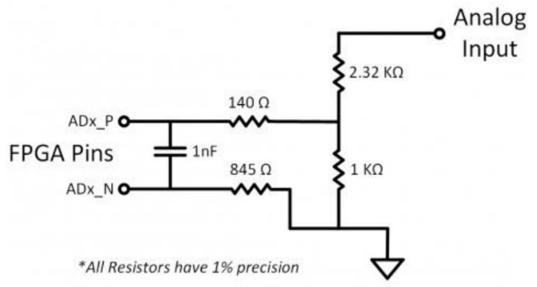
img src: https://www.circuitcrush.com/digital-analog-converters-tutorial/

Interfaces: Analog

- Can we use this on our boards?
  - Yes, Artix-7 has an on-board ADC called XADC on J3
- It's a differential ADC (connect GND to N pin)
- From the reference manual:
  - "The XADC core within the Artix-7 is a dual channel 12-bit analog-to-digital converter capable of operating at 1 MSPS. Either channel can be driven by any of the auxiliary analog input pairs connected to the JXADC header."



img src: https://digilent.com/reference/_media/reference/programmable-logic/basys-3/basys3_rm.pdf



img src: https://digilent.com/reference/_media/reference/programmable-logic/cmod-a7/cmod_a7_rm.pdf

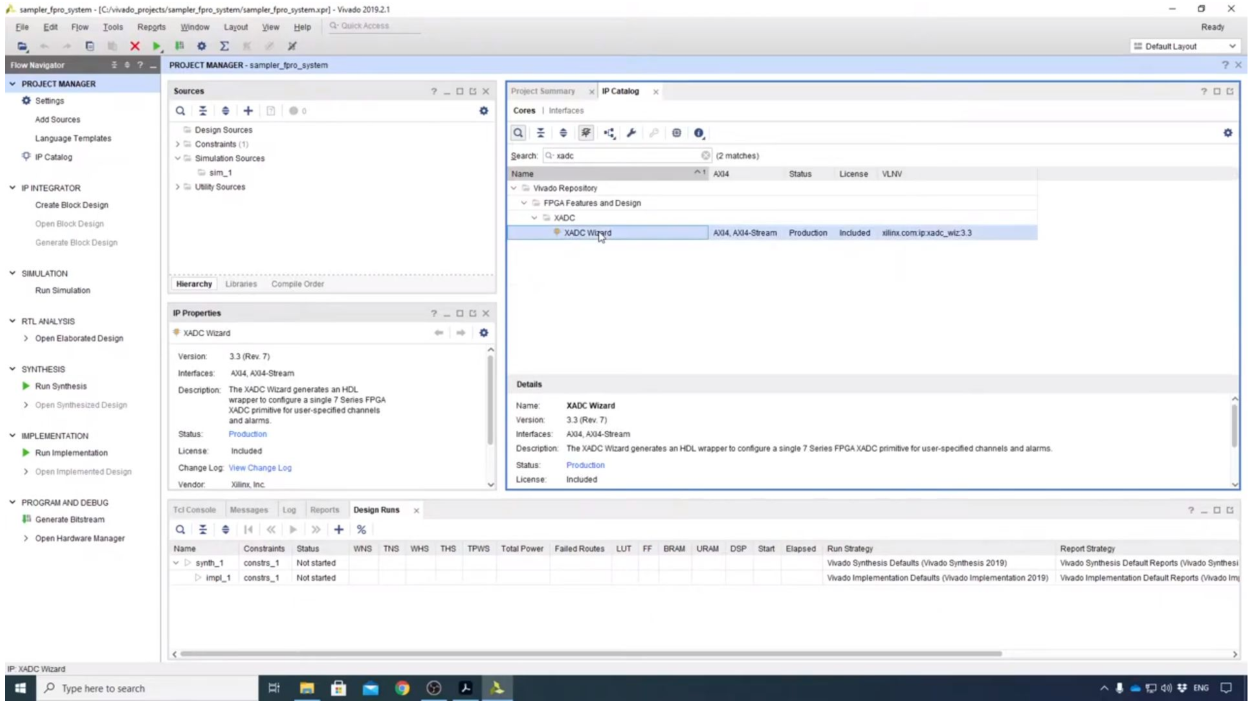
Interfaces: Analog

- XADC is not an RTL module (e.g., debouncer). It’s an IP Core Xilinx provides (e.g., DSP48E1)

- Select it from the IP Catalog and just instantiate it

- We configure the rest of our source files to work **with** it

- I will share a tutorial on this later on. You can use this IP catalog for different cores in your projects



Search: xadc (2 matches)

Name	AXI4	Status	License	VLNV
Vivado Repository				
FPGA Features and Design				
XADC				
XADC Wizard	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:xadc_wiz:3.3

IP Properties

XADC Wizard

Version:	3.3 (Rev. 7)
Interfaces:	AXI4, AXI4-Stream
Description:	The XADC Wizard generates an HDL wrapper to configure a single 7 Series FPGA XADC primitive for user-specified channels and alarms.
Status:	Production
License:	Included
Change Log:	View Change Log
Vendor:	Xilinx, Inc.

Details

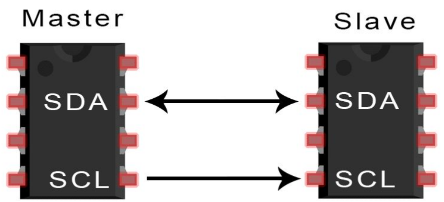
Name:	**XADC Wizard**
Version:	3.3 (Rev. 7)
Interfaces:	AXI4, AXI4-Stream
Description:	The XADC Wizard generates an HDL wrapper to configure a single 7 Series FPGA XADC primitive for user-specified channels and alarms.
Status:	Production
License:	Included

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy	Report Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2019)	Vivado Synthesis Default Reports (Vivado Synthesi
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2019)	Vivado Implementation Default Reports (Vivado Im

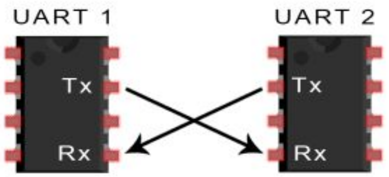
Interfaces: Digital Comms

- Digital communication protocols are perhaps the most widely available modules for FPGAs since they are typically platform-independent (XADC is Xilinx's core)

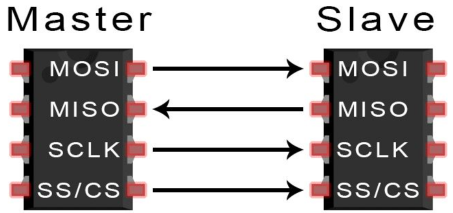
- Examples:
  - I2C:



  - UART:

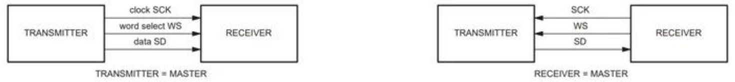


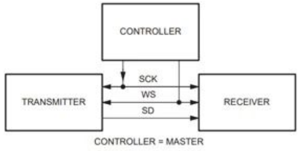
  - SPI:



img src: https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/

  - I2S (audio):





Interfaces: Digital Comms

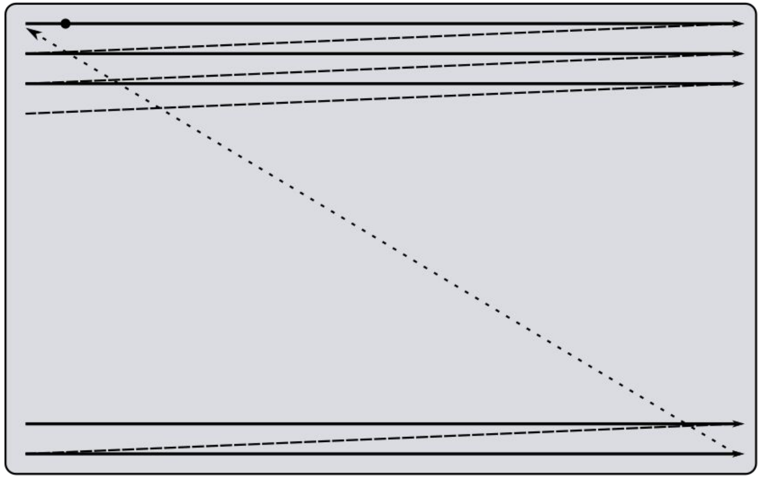
- There are many types, characterized by
  - How many wires they need
  - How they manage collisions (master-slave, ALOHA, …)
  - Reliable speeds / other physical layer characteristics,
  - …
- “Users” (ICs) of these protocols usually have addresses on a bus, and the master prevents collisions from happening by letting different slaves speak
  - unfortunate terminology, but that’s how the protocols are built
- There’s no upper bound in complexity to such interfaces, the ones we saw are the simplest. Check PCIe, USB, MIPI, CSI, Ethernet etc. implementations to see how complex these can get.

Interfaces: Digital Comms

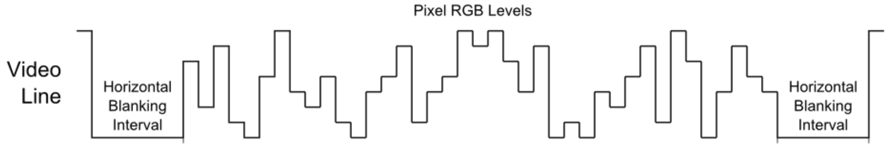
- Many of these interfaces are available in VHDL code, and also specifically for Basys3-compatible modules (e.g., Pmod audio interfaces <-> I2S)
  - https://forum.digikey.com/t/i2c-master-vhdl/12797
  - https://forum.digikey.com/t/i2s-pmod-quick-start-vhdl/13065
  - https://forum.digikey.com/t/uart-vhdl/12670
  - https://forum.digikey.com/t/spi-master-vhdl/12717
- You don’t have to rewrite these in your projects. Just check the documentation to see how they work, use them, cite them, modify them as needed.

Interfaces: Graphics

- Modern day graphics are predominantly digital (HDMI, DisplayPort etc.), but analog graphics drivers are still around (VGA)
- A VGA driver is a very good challenging FPGA project that has an analog counterpart
- VGA relies on something called a raster scan →→→
  - doing zig-zags on the screen and writing out the values for every pixel in 24-bit RGB in the process
- There are DACs on each color channel writing the pixel values on the screen



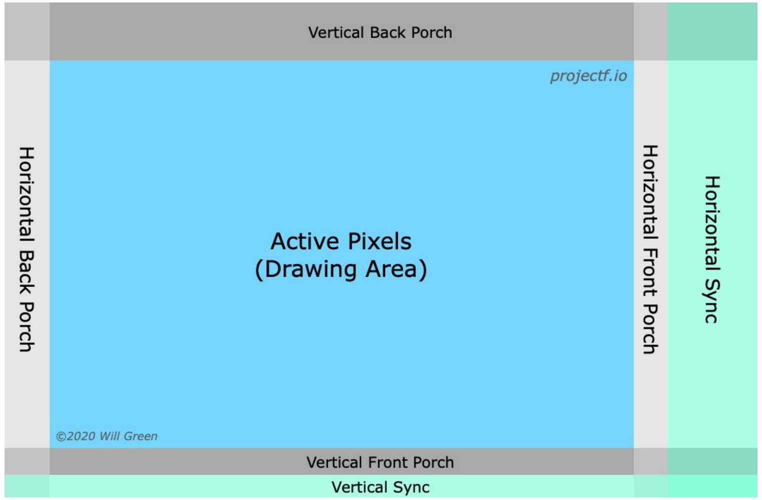
img src: https://projectf.io/posts/fpga-graphics/



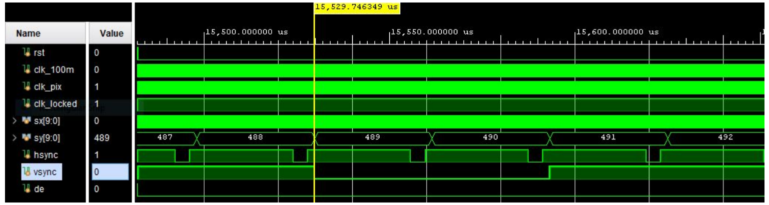
img src: https://electronics.stackexchange.com/questions/166757/can-reading-vga-signals-from-my-computer-harm-the-hardware

Interfaces: Graphics

- There are addressable regions on the signal that correspond to the physical pixels on the screen as well as some “porch” areas that are included for signal processing purposes
- They increase resilience to noise and other artifacts
- The pixel clock is a fixed value, close to 25.1 MHz
- We generate this odd-valued clock via the clk_wiz in Vivado and then get to work on the logic behind
- You can drive VGA displays to do realize arbitrary outputs this way! It’s just another circuit in Vivado

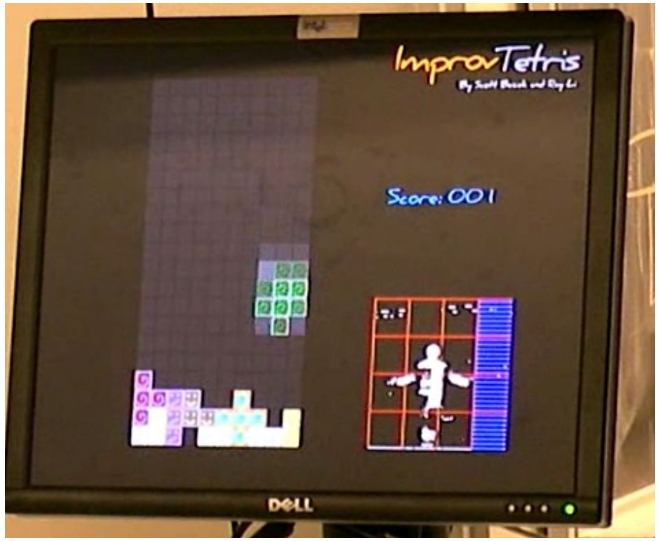


img src: https://projectf.io/posts/fpga-graphics/



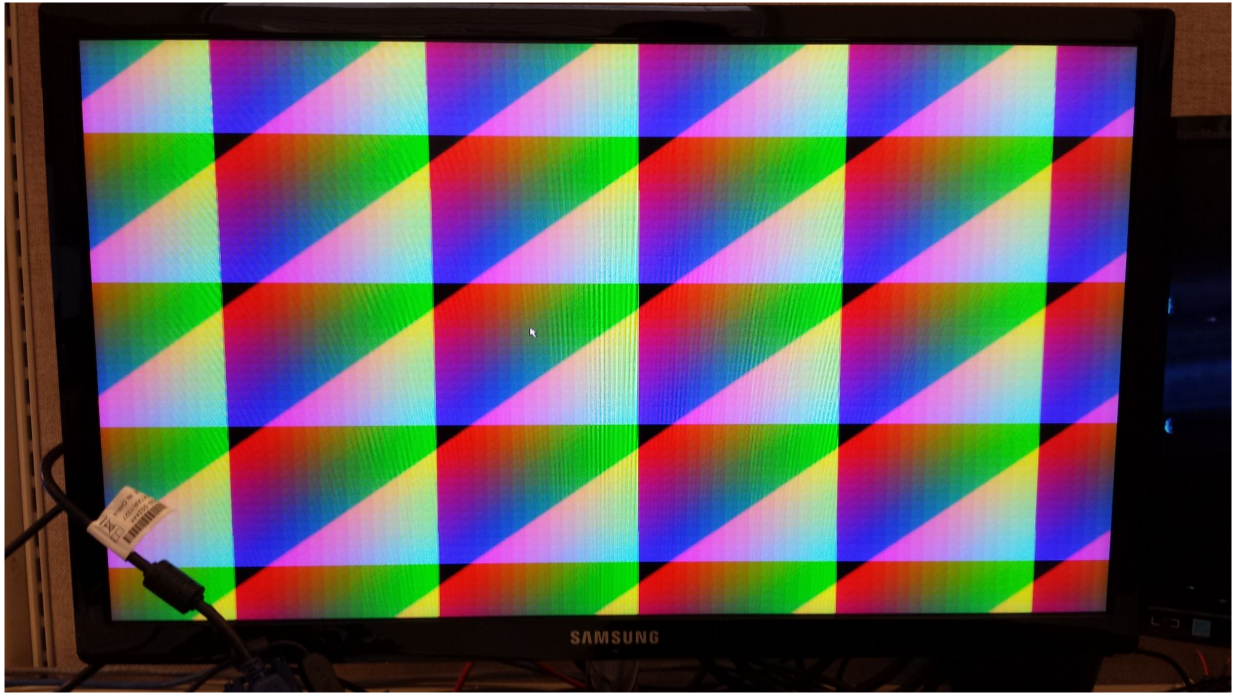
Interfaces: Graphics

- VGA-based games are pretty fun to build and play
- These are practically your own arcade game machines!



Interfaces: Graphics

- Basys3 VGA demo (with a mouse):



Basys3 GPIO Project

- Digilent (the company that buys FPGAs from Xilinx / AMD and builds-sells the Basys3 boards) has an example project for the Basys3 called the “GPIO Project”:
  - https://github.com/Digilent/Basys-3-GPIO

- Don’t let the name fool you, the project is not for just blinking LEDs, it has simple VHDL modules for common interfaces

1) Switches and LEDs
2) Buttons and 7-segment Displays
3) VGA Monitor
4) USB Mouse
5) UART (actually this is a virtual COM port on a PC over USB)

- This is an important resource for your projects, which will surely use at least one of these interfaces

- Let’s have a look at the website and the Vivado project next → ~~neural networks (NNs) on FPGAs~~

~~there’s just one lab left on NNs, and a few short lectures~~

we’ll heavily focus on your projects from now on

🙋🏽‍♂️