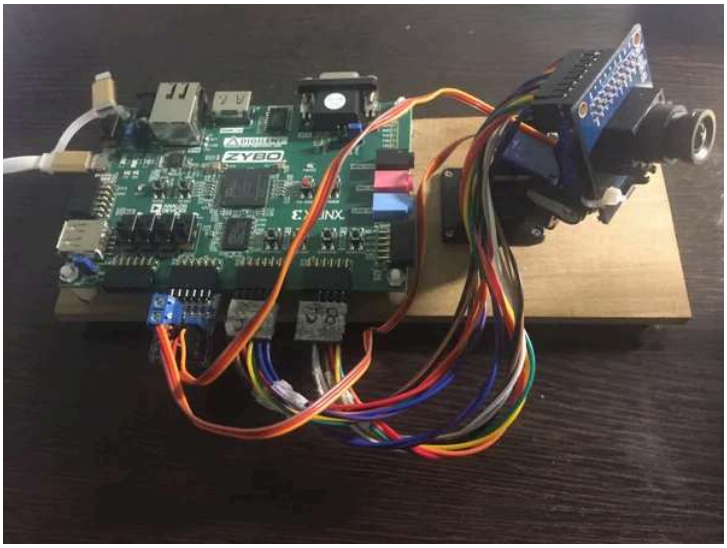
AUTODESK Instructables

# ZYBO OV7670 Camera With Pan/tilt Control

By crazygerbil in CircuitsMicrocontrollers

(cc) BY-NC-SA

## Introduction: ZYBO OV7670 Camera With Pan/tilt Control




Start at step one for detail on just creating an 2-axis servo PWM controller.

Start at the massive block diagram (Step 19) for the full project.

Camera + Pan/tilt setup we used: https://www.amazon.com/gp/product/B013JF9GCA

The PmodCON3 from Digilent was used to connect the servos.

## Step 1: Building a PWM Module- Source File

# Step 2: Building a PWM Module- Vivado Setup


First, download the Vivado Design Suite From Xilinx website. Install all the design suite, including Vivado Software Development Kit(SDK). This project uses 2017.2 version.

In the meantime, Digilent Adept 2 should also be installed as a Zybo board driver.

# Step 3: Building a PWM Module- Create a Project File


Before creating a project file, you should ensure that you've already installed the Zybo file properly as the tutorial here:

Vivado Version 2015.1 and Later Board File Installation

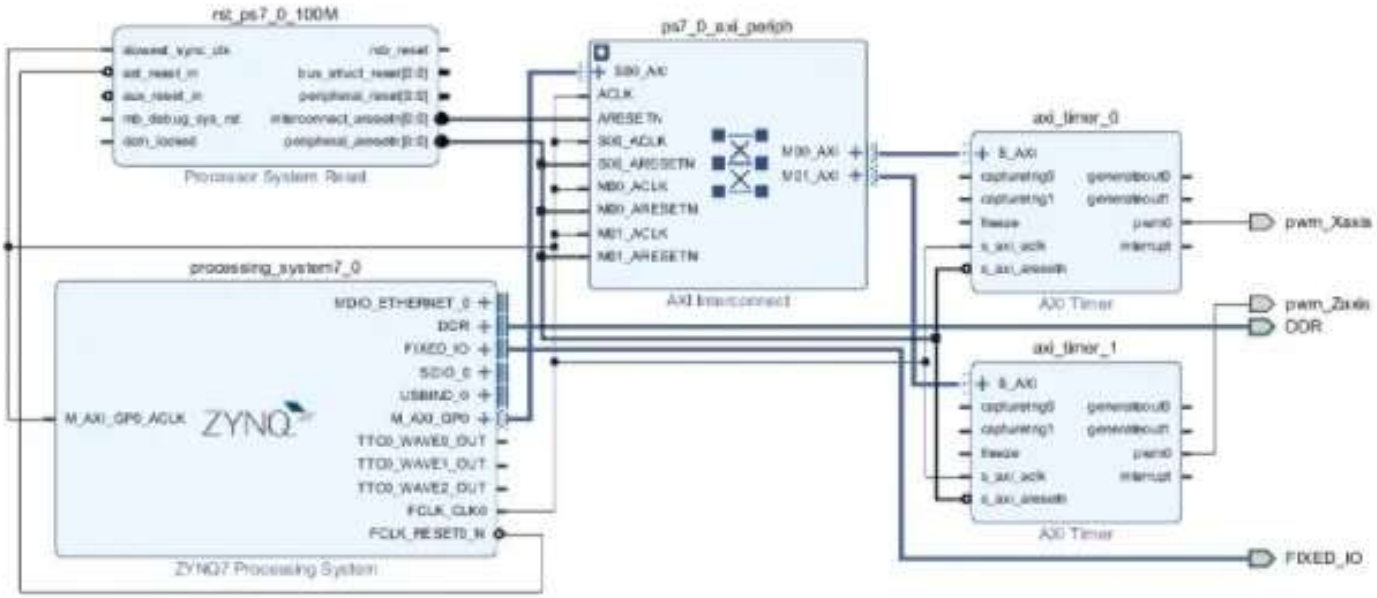
Open Vivado 2017.2. On Quick Start, click on Create Project -> Next -> Project Name(Name your project name here) -> Project Type. On Project Type, select RTL Project and mark on "Do not specify sources at this time". Next, for the Default Part, select "Boards" and "Zybo" as Display Name. Next, Click Finish to start the project.

# Step 4: Building a PWM Module- Block Design and Constraint File Setting (I)


On Flow Navigator, click on ""Create Block Design", then press OK. click the "+" sign to add necessary IPs. Add:

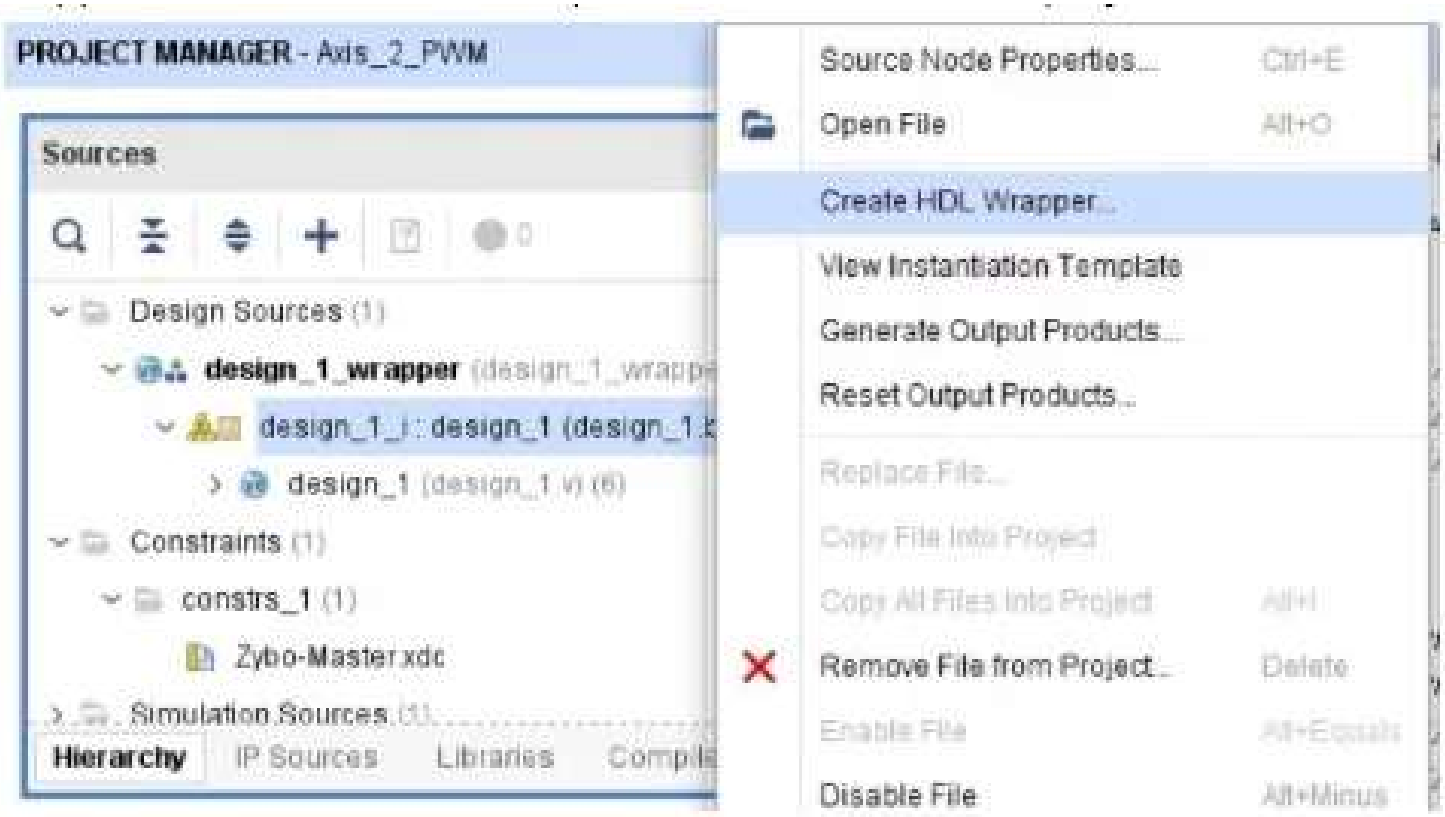
- One ZYNQ7 Processing System Two AXI Timer
- Two AXI Timer

# Step 5: Building a PWM Module- Block Design and Constraint File Setting (II)




After Adding IPs, Run Block Automation and connection automation. Once the automation is completed, on the block “axi_timer_0”, right click on pwm0 -> Make External. Name the pwm0 external pin as pwm_Xaxis. Also, repeat the above process on the block “axi_timer_1” and name the pwm0 external pin as pwm_Zaxis.

# Step 6: Building a PWM Module- Block Design and Constraint File Setting (III)




Notice that every time when we finish the Block Design in Vivado, we need to create an HDL Wrapper. Since it will be the top-level module for each project.

# Step 7: Building a PWM Module- Block Design and Constraint File Setting (IV)

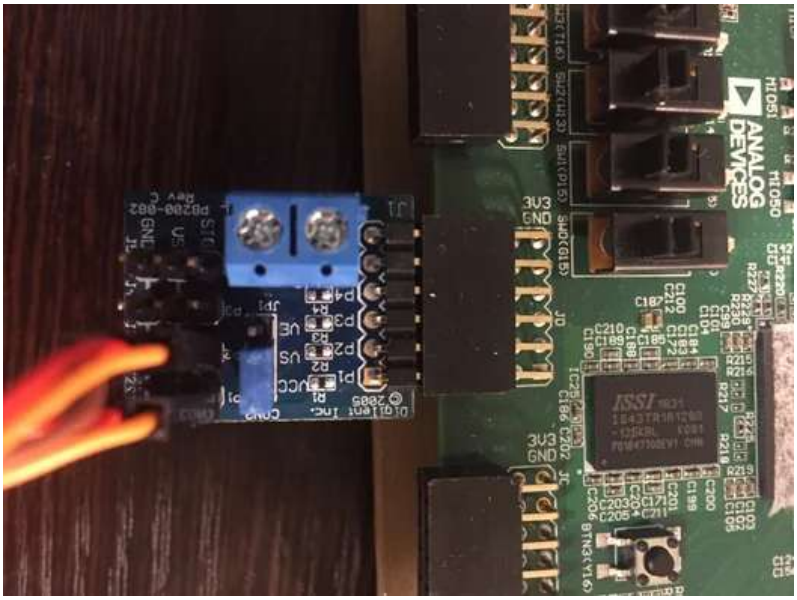

Now, we need to set up our constraint file to assign pins connected to our block diagram. Close the Block Design window, On Sources tab, “Add Sources”->Add or create constraints-> add the Zybo-Master.xdc as our constraint files.

# Step 8: Building a PWM Module- Block Design and Constraint File Setting (V)

```
##Pmod Header JD
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { pwm_Xaxis }]; #IO_L5P_T0_34 Sch=JD1_P
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { pwm_Zaxis }]; #IO_L5N_T0_34 Sch=JD1_N
#set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { id_p[1] }]; #IO_L6P_T0_34 Sch=JD2_P
#set_property -dict { PACKAGE_PIN R14   IOSTANDARD LVCMOS33 } [get_ports { id_n[1] }]; #IO_L6N_T0_VREF_34 Sch=JD2_N
#set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { id_p[2] }]; #IO_L11P_T1_SRCC_34 Sch=JD3_P
#set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports { id_n[2] }]; #IO_L11N_T1_SRCC_34 Sch=JD3_N
#set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { id_p[3] }]; #IO_L21P_T3_DQS_34 Sch=JD4_P
#set_property -dict { PACKAGE_PIN V18   IOSTANDARD LVCMOS33 } [get_ports { id_n[3] }]; #IO_L21N_T3_DQS_34 Sch=JD4_N
```

Open the constraint file Zybo-Master.xdc from Constraints folder, uncomment the ports we want to specify as output signals and rename “ get_ports{XXXX}”, which XXXX denotes the external pin named in the Block Diagram. The setting of constraint file is shown in the figure.

# Step 9: Building a PWM Module- Hardware Installation




Connect the servo motors to the Pmod CON3. TowerPro SG90 is the servo motor model we used in this project. For the servo motor wires, the orange wire represents the PWM signal, connected to SIG pin in Pmod CON3. The red wire Vcc is a power wire connected to VS pin in Pmod CON3. Finally, the brown wire Gnd is a ground wire connected to GND pin. Next, insert the Pmod CON3 to the upper row of JD port in the Zybo Board.

# Step 10: Building a PWM Module- Generate Bitstream and Launch SDK

1.In the Project Navigator tab, run Generate BitStream.

2.Export hardware: File > Export > Export Hardware-> mark on “include bitstream”-> OK 3.Launch SDK : File -> Launch SDK.

# Step 11: Building a PWM Module- Create a New Application in Xilinx SDK


Create a new application :

File > New > Application Project -> Enter the name of your project -> Finish

Under Project Explorer, there should be three folders.

In this case, “design_1_wrapper_hw_platform_0” is the folder previously exported by Vivado. Axis_2_PWM_SDK_bsp is the board support package folder. And Axis_2_PWM_SDK is our main project folder in SDK. You can see “helloworld.c” file under the “src” folder of Axis_2_PWM_SDK, where “helloworld.c” is the main file.

# Step 12: Building a PWM Module- Overview of the Project Explorer (I)

Let’s check some files under Project Explorer. First, in the folder “design_1_wrapper_hw_platform_0”, open the “system.hdf”. This file demonstrates the address map for processor ps7_cortex9 and the IP blocks present in our design.

## Step 13: Building a PWM Module- Overview of the Project Explorer (II)

Then, check the “include” and “libsrc” file under “the Axis_2_PWM_SDK_bsp” folder. Library files here enable us to interact with hardware peripherals without “playing” registers.

## Step 14: Building a PWM Module- Overview of the Project Explorer (III)

Through the BSP documentation, xtmrctr.h is found as a Xilinx Timer Control Library related to AXI Timer. Typically, we could find the desired PWM function here. However, if you read the documentation “tmrctr_v4_3”, it shows that the driver doesn’t currently support the PWM operation of the device. Due to the deficiency in PWM function, we have to wrap up our PWM function with the aid of xtmrctr.h and AXI Timer v2.0 LogiCORE IP Product Guide .

## Step 15: Building a PWM Module- Wrap Up PWM Function (I)

Back to the main file “helloworld.c”, include the following header files:

## Step 16: Building a PWM Module- Wrap Up PWM Function (II)

Define the base addresses of two AXI TImer through “xparameters.h”.

## Step 17: Building a PWM Module- Wrap Up PWM Function (III)

Build the desired PWM function.

**Duty_val**: converts the degree value into duty cycle.
**PWM_Freq_Duty**: set the desired frequency and duty cycle to generate PWM. Clock period should also be assigned.

**PWM_START**: assign the PWM register address and start generating PWM.

**PWM_STOP**: assign the PWM register address and stop generating PWM.

The rest of demo code is shown in “helloworld.c” under “Axis_2_PWM_SDK”

# Step 18: Building a PWM Module- Make It Run!

1.Program the FPGA through SDK

- Connect Zybo Board through USB port to PC.
- Xilinx Tools -> Program FPGA

2.Run the program

- Click on “Run” icon and drop down the menu -> Run As -> Launch on Hardware

3.SDK Terminal

- Open the SDK Terminal -> Connect to Serial Port -> OK
- Run the program. If the demo code runs successfully, you should see “Initialization done!” on the SDK Terminal.

# Step 19: Streaming Video Processing on Digilent ZYBO With OV7670

Complete archive file attached.

# Step 20: Complete Block Diagram

This shows the complete diagram of all connections and IP blocks in the project

# Step 21: Connect the OV7670 to the ZYBO

Create a connection to wire the ov7670 module to the ZYBO Pmods

Data Pmod is Pmod D

Control Pmod is Pmod C

Additionally, connect the PmodCON3 and servos as specified in the first half of this tutorial

# Step 22: Create Block Design

In Flow Navigator, click on "Create Block Design", then press OK.

# Step 23: Add VHDL Files for OV7670 Camera Control and Capture

Add the VHDL files attached to this step to the project

# Step 24: Add Constraints File

Add the attached constraints file to your project.

# Step 25: Add IP Repo for HLS IP

Take the Zip file attached and **unzip it** into a new folder named similarly in a new directory (folder) called "HLS_repo".

Add an IP repository to your project by going to the IP catalog and right-click selecting "Add Repository..."

Navigate to your "HLS_repo" directory and select it.

Optional: Create the HLS video processing block for yourself!

# Step 26: Add Modules and IP

Add the ov7670_axi_stream_capture, debounce and ov7670_controller modules to the block diagram by right clicking the background and selecting "Add Module..."

Similarly, add the IPs:

- HLS_Video_Track
- Video Frame Buffer Write
- Video Frame Buffer Read
- Video Timing controller
- AXI4-Stream to Video Out
- 3 of "Slice"
- Constant
- 2 of AXI Timer

# Step 27: IP Configuration Settings

As shown in pictures

# Step 28: Add and Configure PS IP Block

Add the ZYNQ7 Processing System to the block diagram

edit the configuration:

- PS-PL configuration
  - HP
    - Enable S HP 0
    - Enable S HP 1
- Clock configuartion
  - PL Fabric Clocks
    - FCLK_0 at 100MHz
    - FCLK_1 at 25MHz (OutputClock)
    - FLCK_2 at 35MHz (<=50MHz) (CameraClock)

# Step 29: Part 1. Building a PWM Module for Servo Motors

Bring out axi_timer_0 pwm0 to new output port pwm_Xaxis

Bring out axi_timer_1 pwm0 to new output port pwm_Zaxis

# Step 30: Video Input Side Connections (aclk Highlighted)

Connect the video input side IP blocks correctly

(* these connections should be created by selecting the right options during connection automation)
"aclk" from the axi_stream_capture goes to:

- ap_clk on video frame buffer write
- ap_clk on HLS video stream processing block
- *aclk on AXI smartconnect IP from the Video Frame Buffer Write to the S_AXI_HP0
- *aclk corresponding to the channels of the AXI Interconnect IP for the S_AXI channels of the HLS video processing block and the video frame buffer write S_AXI_HP0_ACLK on the PS block

The video stream signal is simply connected in series from the capture block to the Zynq memory interface.

- Video goes from capture block to HLS processing block.
- The processed video from the HLS block goes to the frame buffer write block.
- *The frame buffer write block connects to the HP0 interface on the Zynq PS block.
- The m_axis_tuser signal from the output of the capture block is manually wired to both the video_in_TUSER input signal on the HLS processing block and the ap_start signal on the same block.

  *The TUSER(tuser) signal is used by the AXI video stream protocol to indicate the start of a frame of the video.*
  *AP_Start tells the HLS block to start processing.*
  *We are thus using tuser to trigger the HLS block to process each frame as it comes in.*
  *When connecting a single signal of a bus and splitting it off in this fashion it is necessary to also connect it the normal termination point of the rest of the bus. Vivado assumes that if you are connecting the signal manually that you want to disconnect what it would normally connect to.*

Configuration settings of IP blocks:

- Video frame buffer write:

```
Video formats: RGB8
1 sample per clock
Max columns: 1280 (>=640)
Max rows: 960 (>=480)
Max data width: 8<br>
```

# Step 31: Connections to OV7670

On the ov7670_axi_stream_capture block

- Make all of the inputs external (right click on a pin and select from the menu, or left click->ctrl+T)
- Leave the names as they are

On the ov7670_controller block

- Make all of the block outputs external
- Rename the config_finished port to led0
- connect clk to the CameraClock (<=50MHz) (FCLK_2)

On the debounce block

- connect the button1 input to an external input port called btn0
- connect the out1 to the resend line on the ov7670_controller IP block
- connect the button2 input to an external input port called btn3
- connect the out2n to the ext_reset_in input on the Processor System Reset IP for the video capture clock domain. (*This may need to be done after that IP is generated*)
- connect clk to the CameraClock (<=50MHz) (FCLK_2)

# Step 32: Connections on Video Out Side

Connections for blocks Video Timing Controller (VTC), AXI4-Stream to Video Out, and slices

- Use 25MHz clock (FCLK_1) for vid_io_out_clk and VTC clk
- Use 100MHz clock (FCLK_0) for aclk on AXI4-Stream to Video Out
- vtiming_out to vtiming_in
- Video Frame Buffer Read m_axis_video goes to AXI4-Stream to Video Out video_in
- vtg_ce goes to gen_clken
- Tie VTC clken, aclken, vid_io_out_ce to Constant dout[0:0]
- Bring vid_hsync and vid_vsync out to external output ports vga_hs and vga_vs, respectively. (not pictured)

Slices:

- Slices should be setup as shown in attached pictures
  - rename the blocks to slice_red, slice_green, and slice_blue
  - setup slice ranges as shown in the images per the name of the block
  - connect each slice output to a external port output as shown in image.
- vid_data[23:0] connects to the inputs for each slice (Din[23:0])

# Step 33: Run Block and Connection Automation

Run Block Automation to connect the things from the ZYNQ7 PS block. As shown by picture.

Run Connection automation to create all interconnection IP. Pay close attention to all options in each picture.

On the debounce block, connect out2n to the video capture clock domain Processor System Reset ext_reset_in input.

# Step 34: Generate HDL Wrapper

Generate the HDL Wrapper for your block design.

Set it as the top module.

# Step 35: Generate Bitstream, Export Hardware to SDK, Launch SDK From Vivado

Make sure to include bitstream in export.

Generation of bitstream may take a very long time.

Then launch SDK

# Step 36: SDK (no FreeRTOS)

This version does everything without using FreeRTOS, condensing the code nicely.

Create a standalone BSP based on the hardware design. The default options should be fine. Make sure BSP sources have been generated.

Create App as shown in Picture. (empty application)

Delete the autogenerated main and import the attached files.

# Step 37: FreeRTOS Implementation

This version does uses FreeRTOS.
Create a FreeRTOS901 BSP based on the hardware design. The default options should be fine. Make sure BSP sources have been generated.

Create App as shown in Picture. (empty application)

Delete the autogenerated main and import the attached files.

# Step 38: Usage Instructions

This project is a bit tricky to get running. Follow the steps in order.

Make sure that your ZYBO is not self-loading anything when it gets powered on. This means the the Done LED should not light up. One way to do this is to set the boot source jumper to JTAG.

Open the project (FreeRTOS or not) you want to program from SDK

1. Turn on your ZYBO. The Done LED should not light up.
2. Program the FPGA with the bit file. The Done LED should light up. Led0 should not light up.
3. Run the code (remember to go past the start breakpoint if you are doing that).

At this point you should be getting an output on your VGA display.

To restart (if it bugs or whatever): quickly tap the PS-SRST button or turn the ZYBO off and then back on. Continue from step 2.

Halting the processor with the debugger will cause the Camera to hold position instead of moving. The video stream will continue anyway.

# Step 39: References and Links

Xilinx reference guides and documentation:

- PG044 - AXI-Stream to Video Out
- PG278 - Video Frame Buffer Read/Write

Other links:

- Lauri's blog - VDMA input
- Lauri's blog - OV7670 to VGA output using BRAM
- Hamsterworks wiki, by Mike Fields, the original source of the OV7670 code
- Datasheet showing basic timing specifications