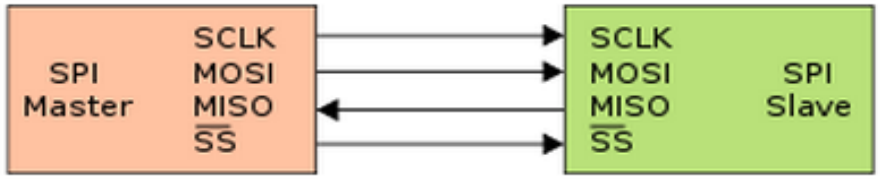
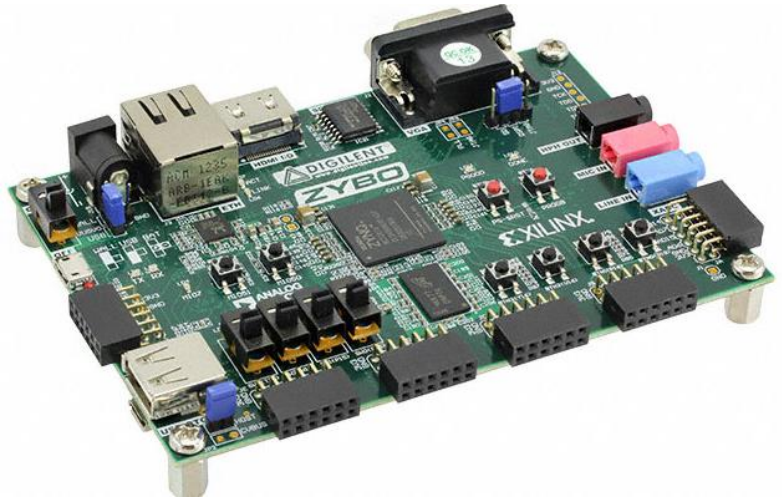
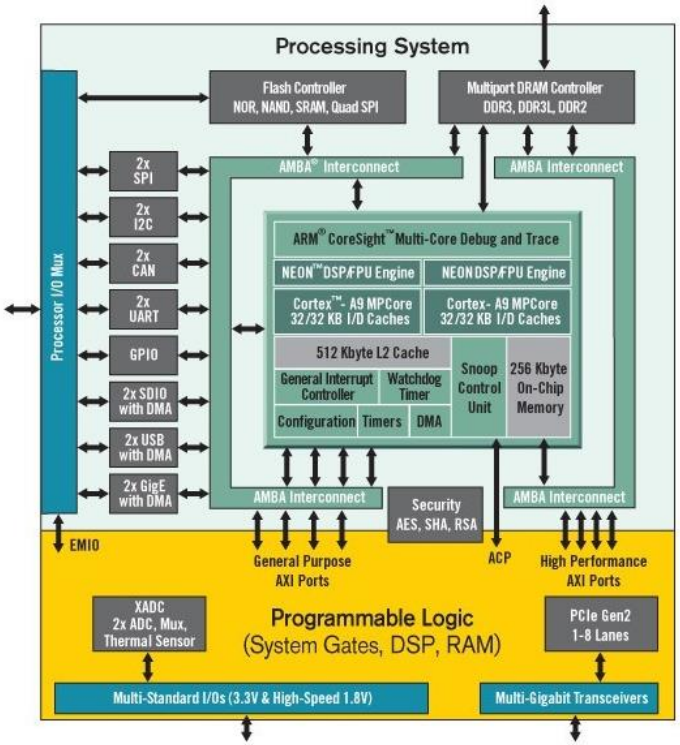
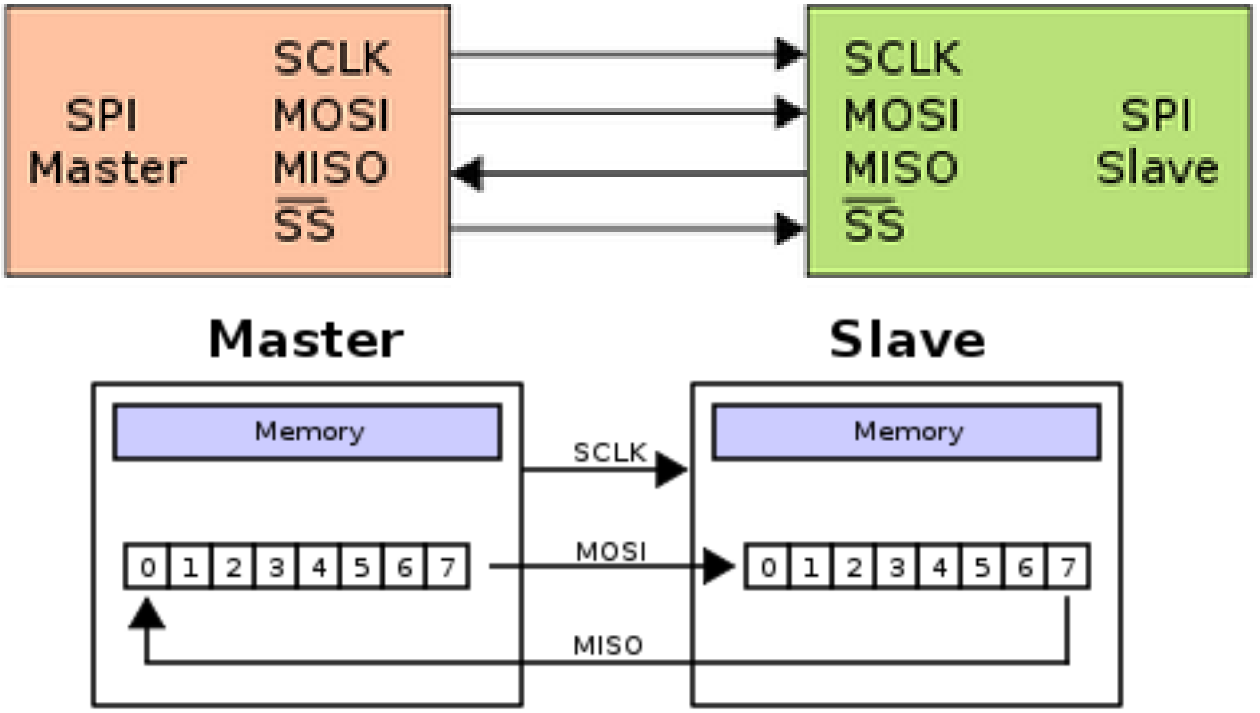
Zynq SPI Peripherals



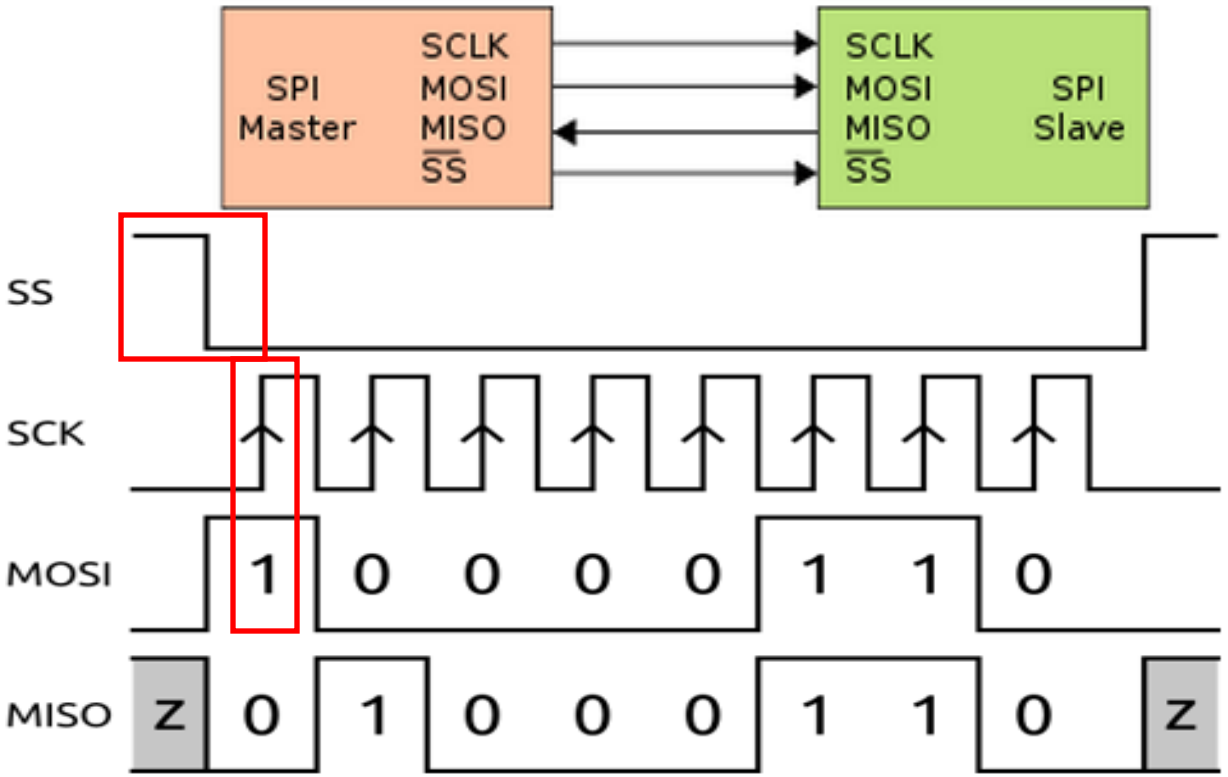




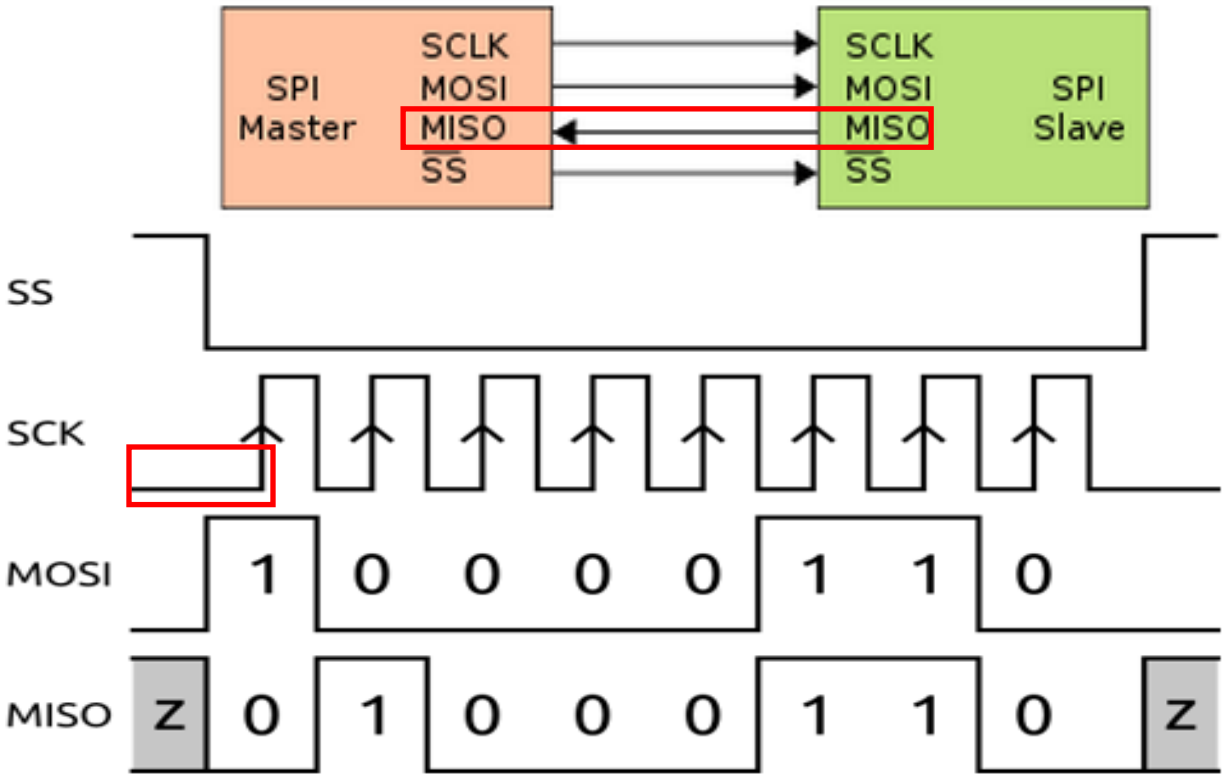
SPI Bus. The *Serial Peripheral Interface* (*SPI*) consists of the serial clock *SCLK*, master output/slave input *MOSI*, master input/slave output *MISO* and slave select *SS* signals.



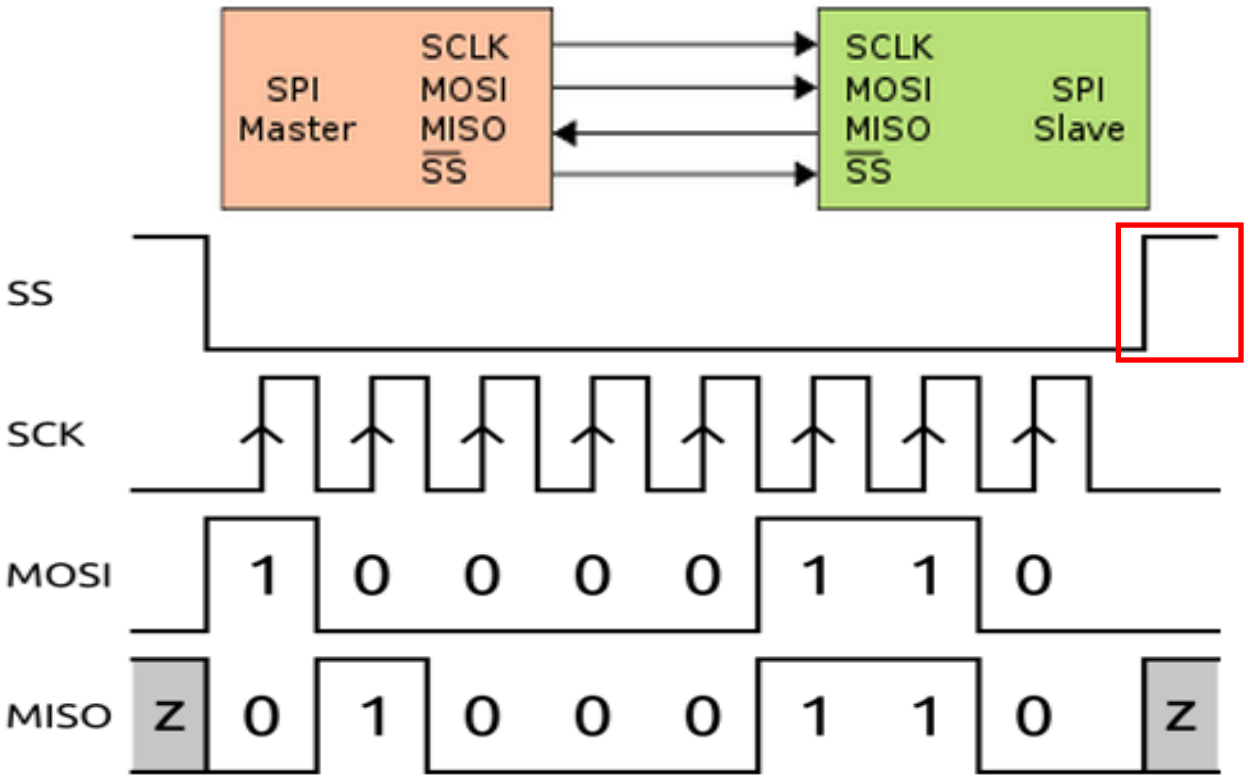
SPI Bus. The SPI bus serial data transmission begins with the SS signal at logic 0. Either the *rising* (as shown) or *falling* edge of SCLK as specified by the slave device signifies valid MOSI data with the *most significant bit* (MSB) sent first.



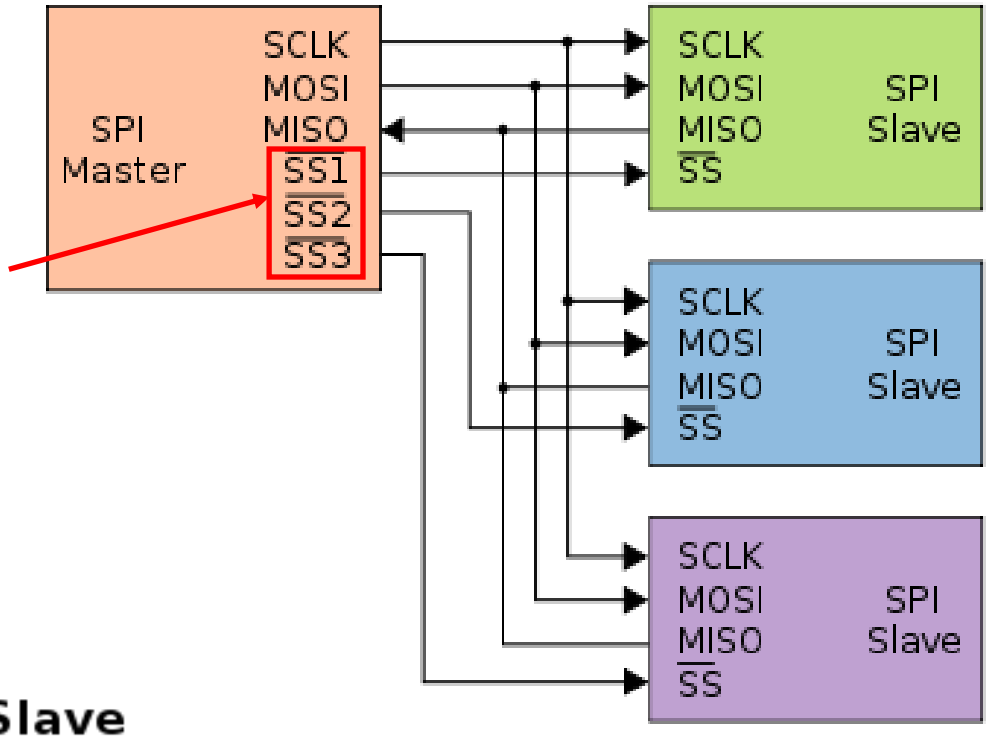
SPI Bus. The default value of the SCLK signal for a rising edge valid data transmission is logic 0 and for a falling edge it is logic 1. Concurrently, the MISO signal outputs the MSB of the *last* (not the current) data transmission to the SPI bus master device.

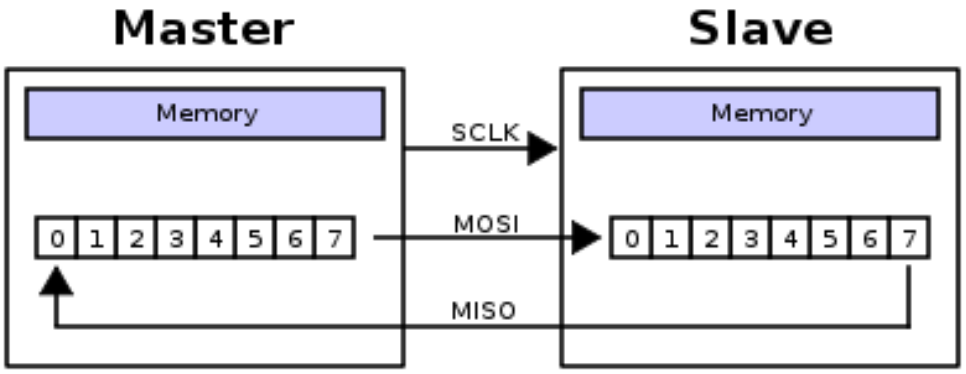


SPI Bus. This serial data transmission continues with any number of SCLK cycles until the SS signal returns to logic 1.

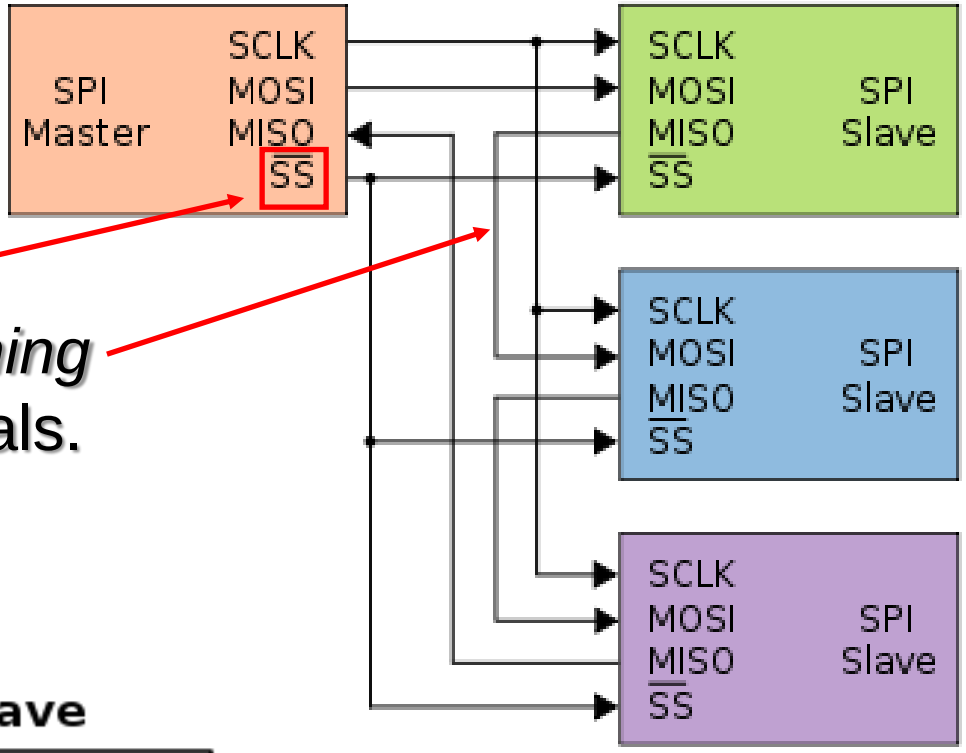


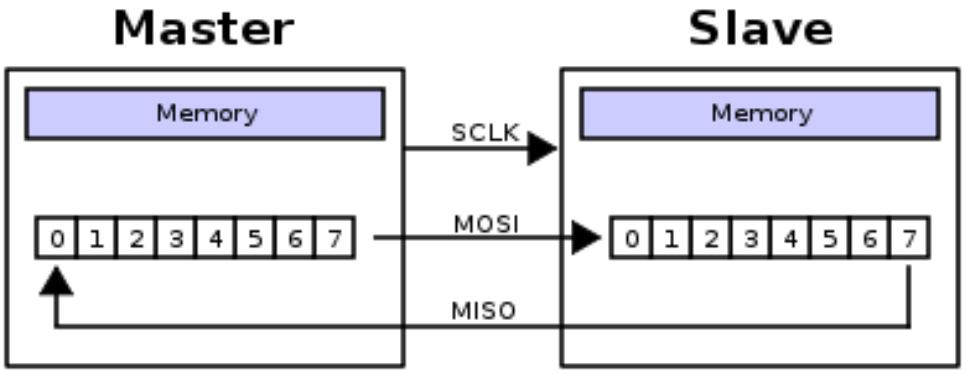
SPI Bus. Multiple independent SPI slave devices can be interfaced to a single SPI master with *multiple* SS signals.



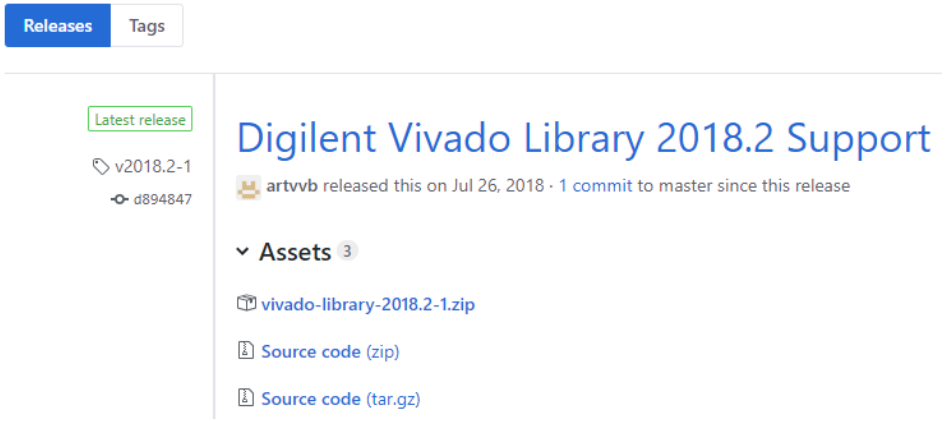


SPI Bus. Alternatively, Multiple cooperative SPI slave devices can be interfaced to a single SPI master with a *single* SS signal and *daisy chaining* the MISO and MOSI signals.





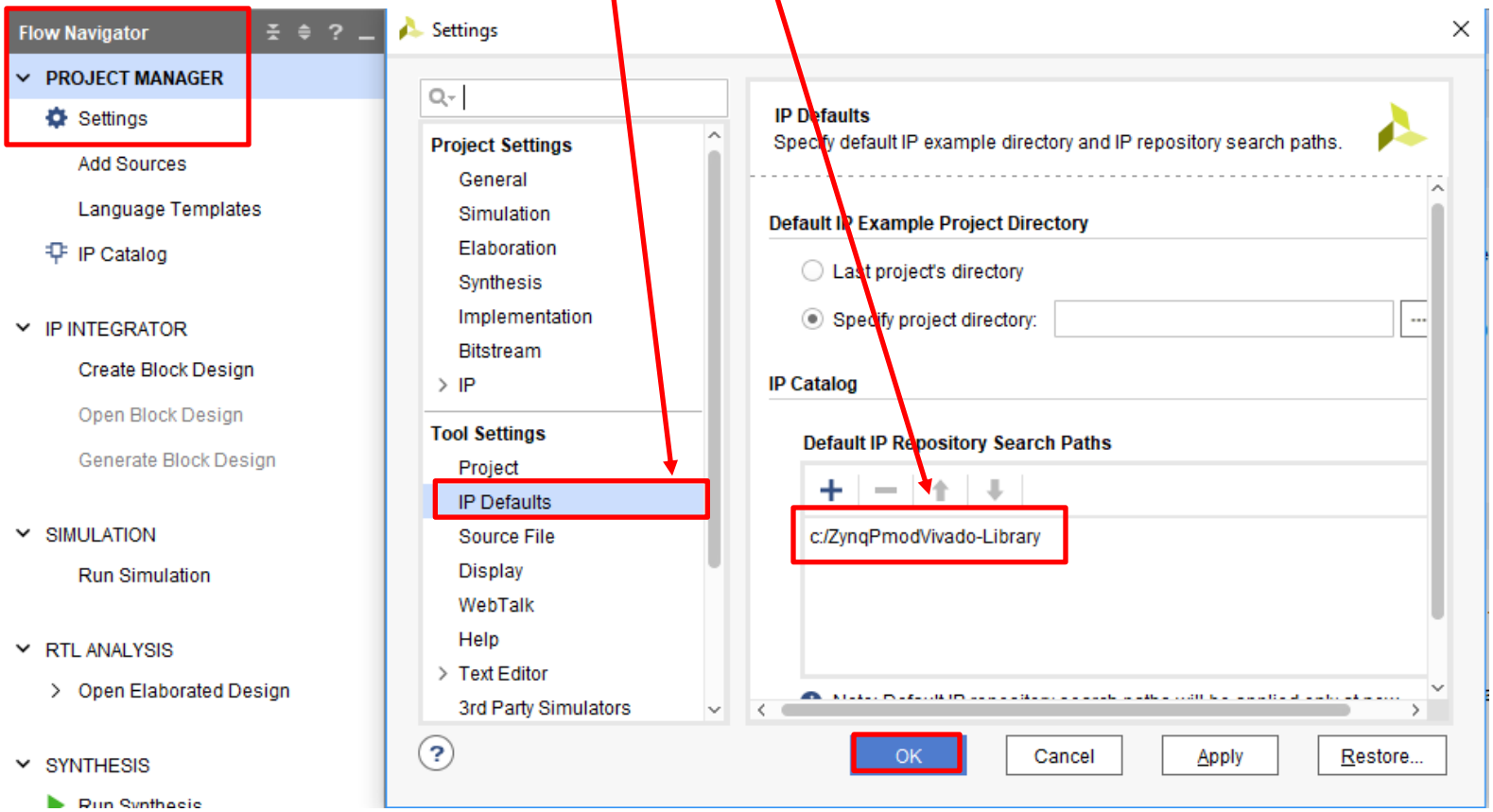
Zynq SPI Peripherals – Vivado Library. Download the latest release of the Digilent IP library at: *https://github.com/Digilent/vivado-library/releases* in zip archive format to a new folder *C:/ZynqPmodVivado-Library*. Extract the archive.



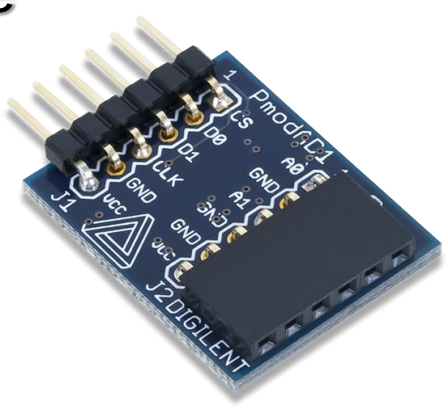
Create a dummy Vivado project as *project_1* with no sources.

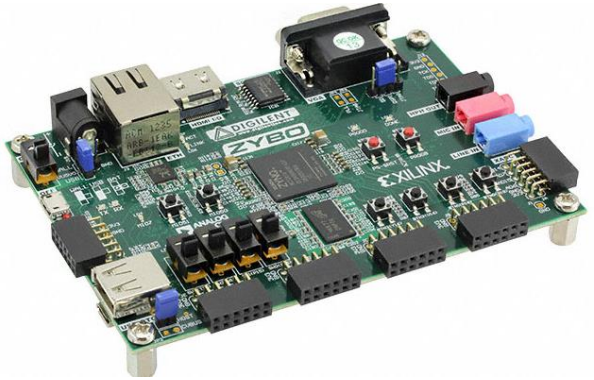
XILINX® VIVADO™

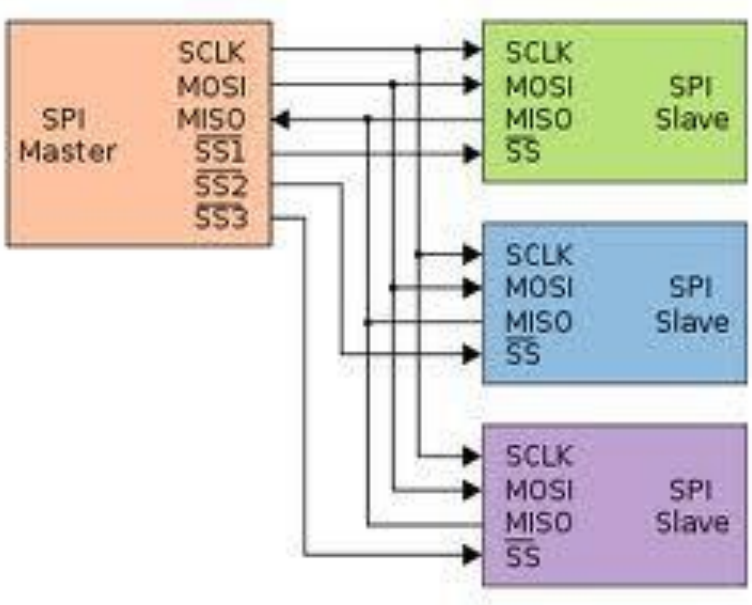
Zynq SPI Peripherals – Vivado Library. In the *Flow Navigator…Project Manager* select *Settings* then *IP Defaults*. Add *C:/ZynqPmodVivado-Library* to the *Default IP Repository*. *OK* to continue.



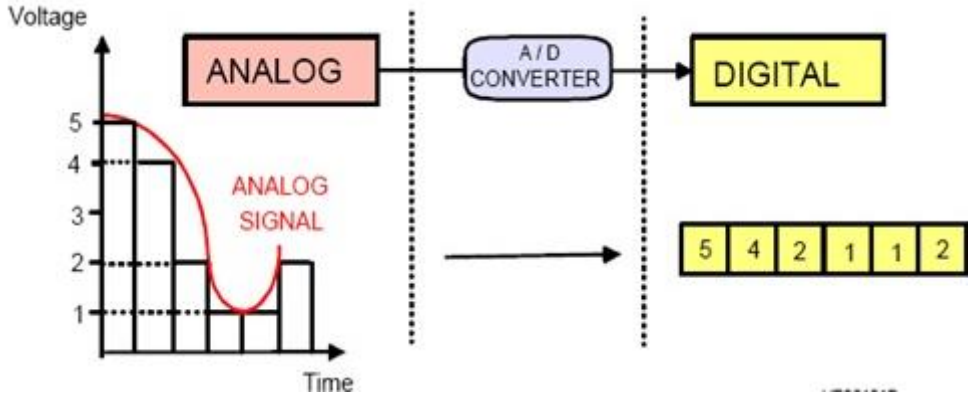
Zynq SPI Peripheral - ADC. As an example of SPI bus interfacing to a Zynq device, a Digilent PmodAD1 two-channel,12-bit resolution analog-to-digital converter (ADC) is interfaced to the Zynq PS in the Vivado HLx design project using the *ZynqPmodVivado-Library*.

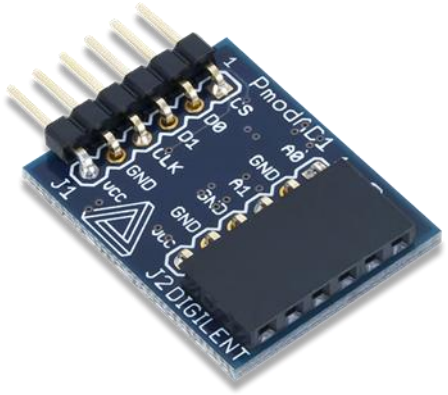






Zynq SPI Peripheral - ADC. An ADC provides a discrete, sampled binary data output signal from a continuous analog input signal for embedded system design in audio processing, analog and digital baseband and bandpass communication and digital process control.

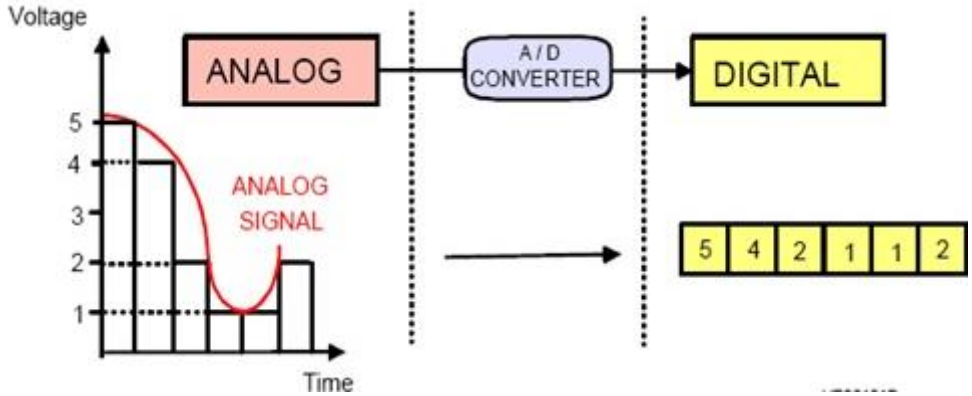


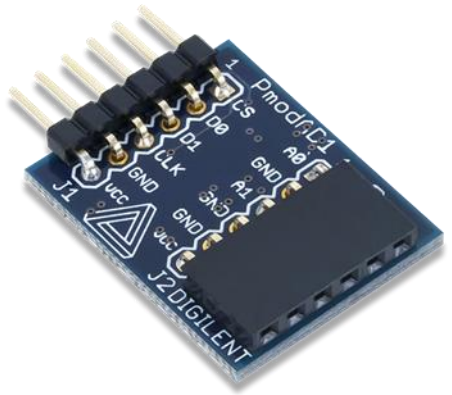


Zynq SPI Peripheral - ADC. The general equation for an n-bit binary data output D[n-1:0] for the ADC that inputs a unipolar (only positive amplitude) analog signal is:

$$D[n-1:0] = G\frac{V_{IN} - V_{REF}}{V_{FS}}2^{n-1}$$

The maximum rate at which the ADC can sample an analog signal and produce n-bit binary data is its throughput rate R_{ADC} in samples per second.

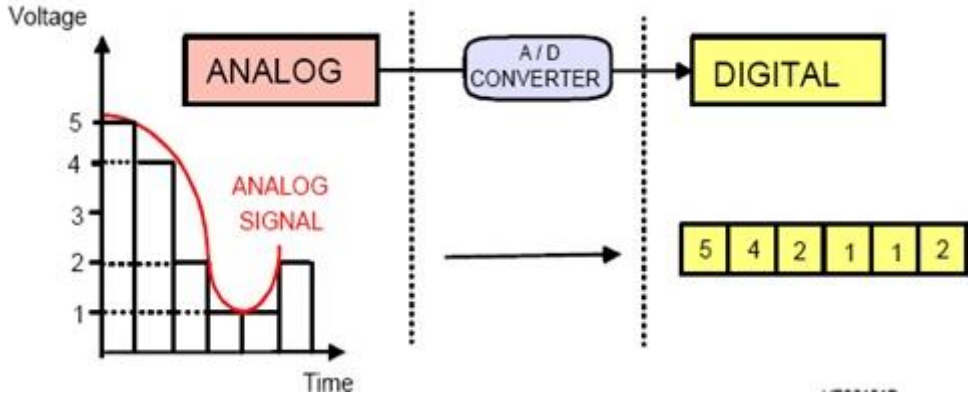


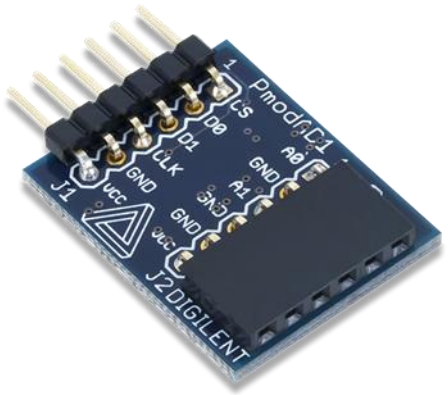


Zynq SPI Peripheral - ADC. Since V_{IN} could be less than V_{REF}, the n-bit binary data output D[n-1:0] is then in *two's complement* format with the most significant bit (MSB) representing the sign bit and a number whose integer value is between -2^{n-1} and $2^{n-1} - 1$

For example is n = 12, then the range is –2048 to 2047.

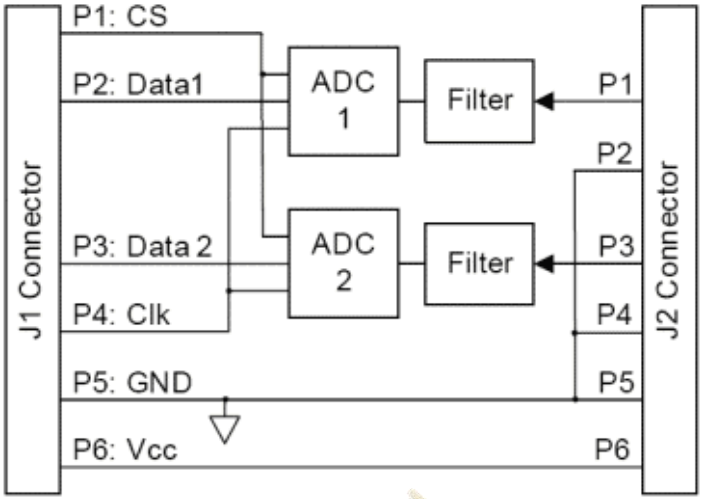
$$D[n-1:0] = G\,\frac{V_{IN} - V_{REF}}{V_{FS}}\,2^{n-1}$$



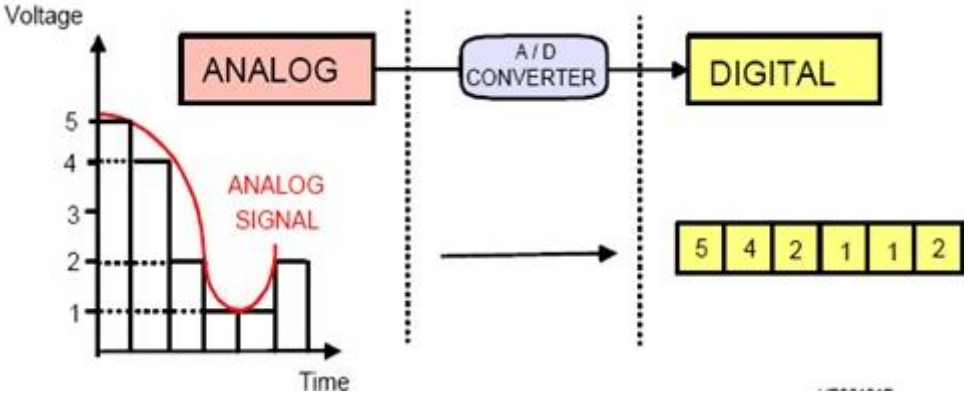


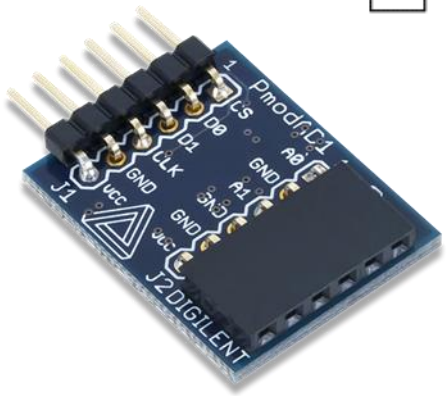
Zynq SPI Peripheral - ADC. However, the PmodAD1 ADC uses G = 1, V_{REF} = 0, V_{FS} = V_{DD} = 3.3 V and n =12 for a digital range of 0 to 4095 for an input voltage for 0 to 3.3 V.

$$D[n-1:0] = G\frac{V_{IN} - V_{REF}}{V_{FS}}2^{n-1}$$

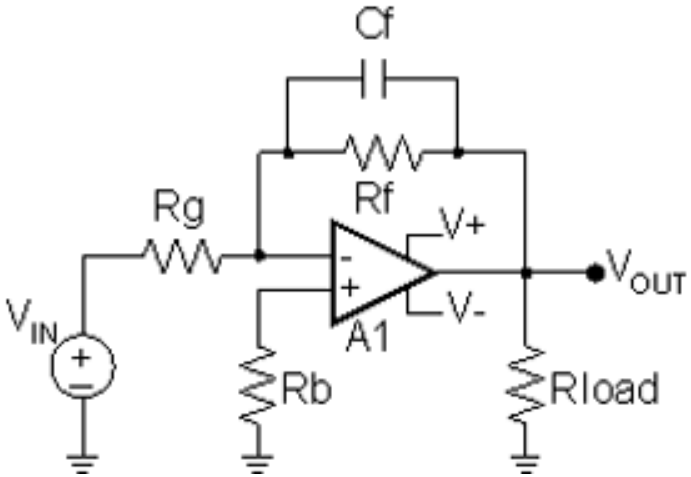


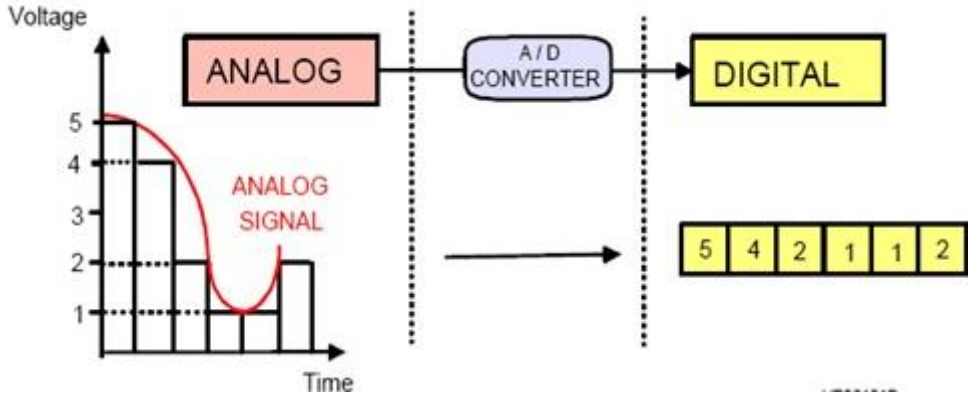
The PmodAD1 has two TI ADCS7476 12-bit ADCs.

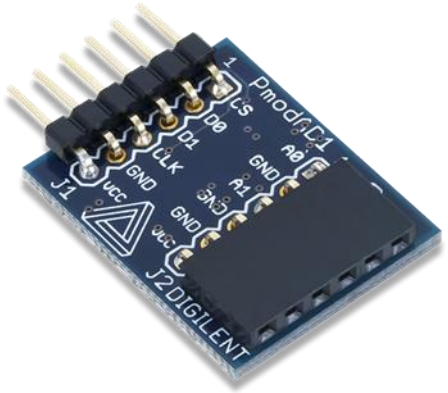




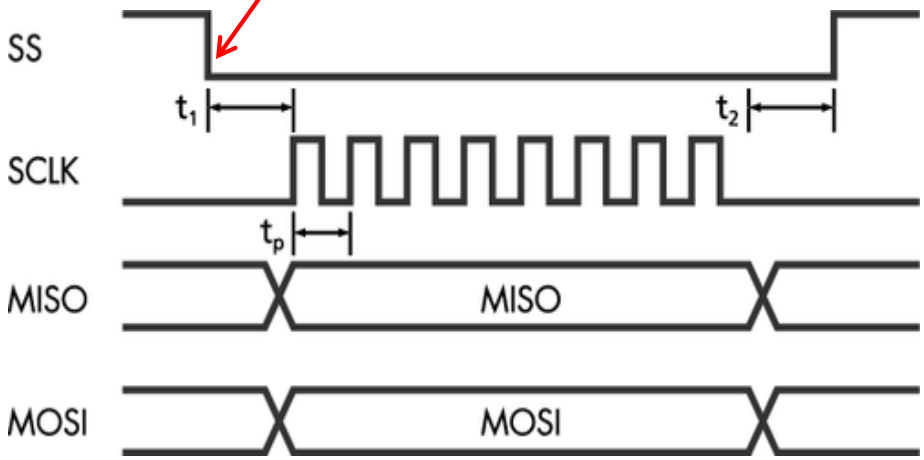
Zynq SPI Peripheral - ADC. An external analog preamplifier can also support the function of the ADC in an embedded design by providing a differential to single-ended signal interface, significant gain for an analog transducer or an *anti-aliasing* filter for discrete data sampling.

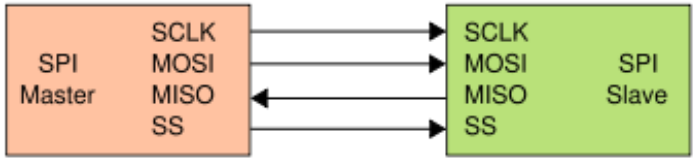


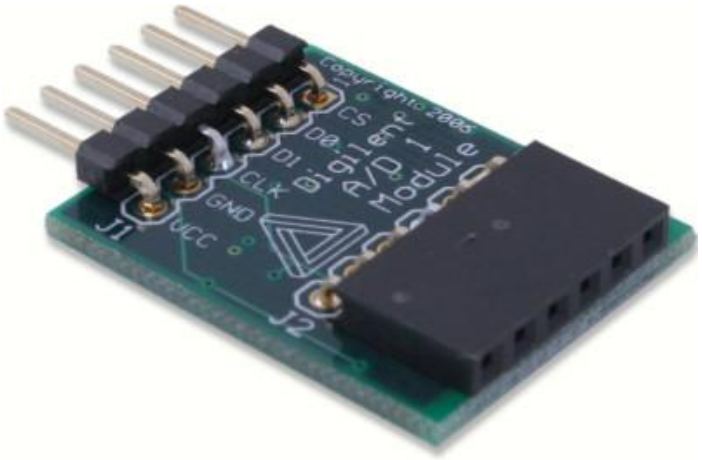




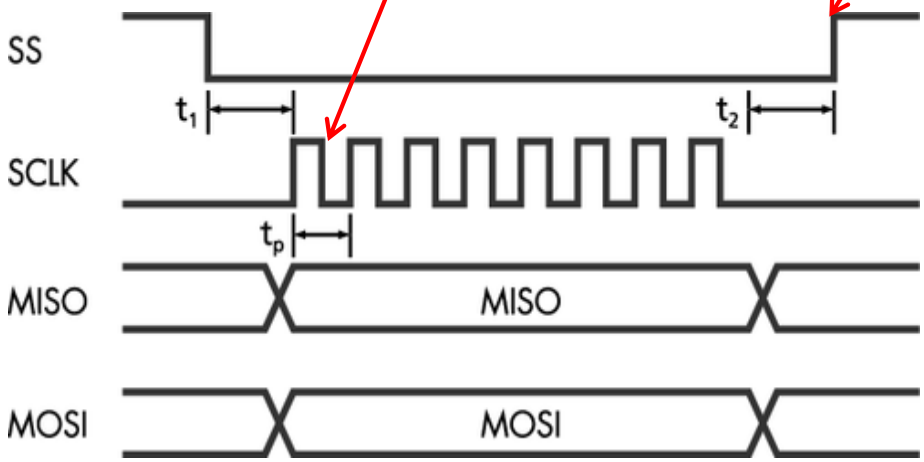
Zynq SPI Peripheral - ADC. The ADCs use an active low ADCCS signal and an SPI bus clock signal SCLK and output two data signals ADCD0 and ADCD1. The ADC SPI data communication protocol begins with the chip select signal ADCCS set to logic 0 which initiates analog data conversion.

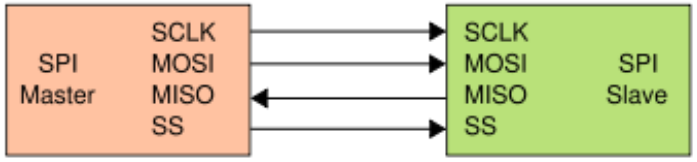


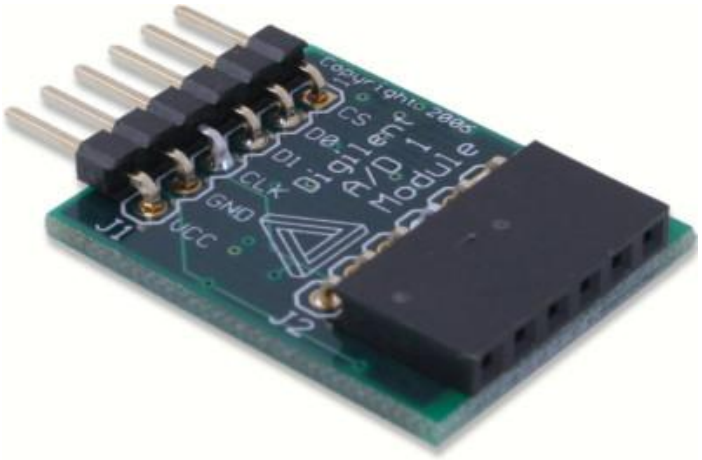




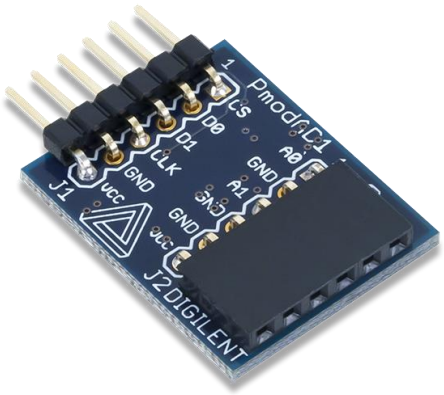
Zynq SPI Peripheral - ADC. The PmodAD1 ADCD7476 ADC uses the falling edge of the SPI clock signal SCLK to transmit a 16-bit data packet, with the first four bits as don't care bits and the most significant bit (MSB) of the data sent first. After all 16 bits have been sent, the chip select signal ADCCS is set to logic 1.



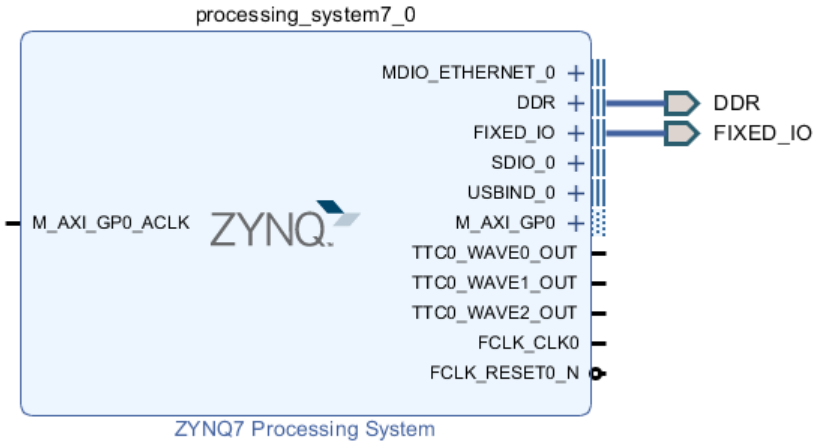




Zynq SPI Peripheral - ADC. As in the Zynq Book Tutorial Exercises, create a Vivado project as *zybo_PmodAD1* in a new folder C:/zybo_Pmod_Projects.

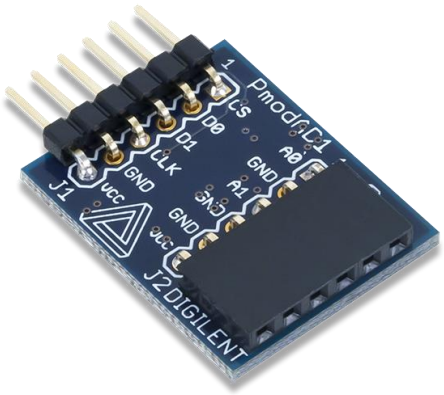


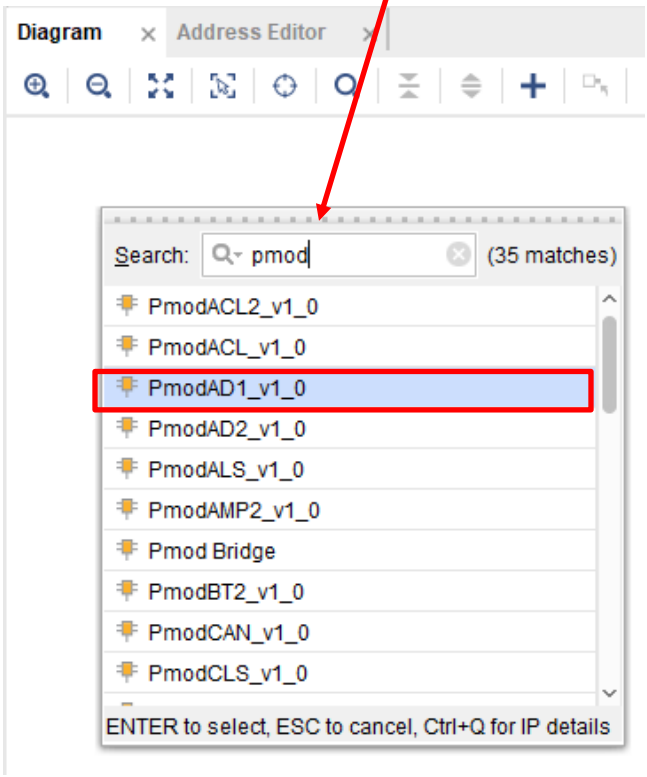
The project is *RTL* and has no source or constraint files. Select Zybo board and configure a basic Zynq processing system. In *Project Manager*…*Settings* the top module name is *design_1*.

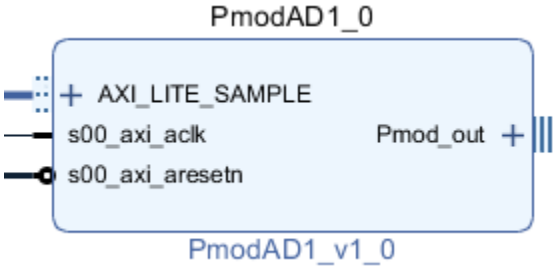


VIVADO™

Zynq SPI Peripheral - ADC. Add IP (**+**) and search for *pmod*. Select *Pmod_AD1_v1_0*. The PmodAD1 does not require a *reference clock* or the servicing of an *interrupt*.

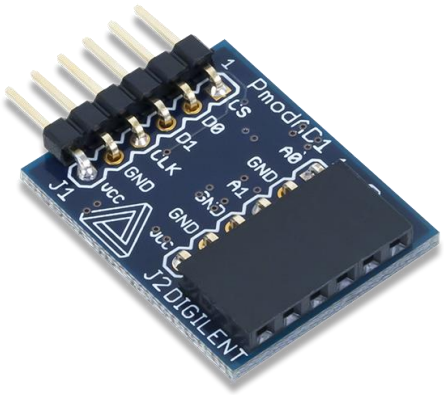


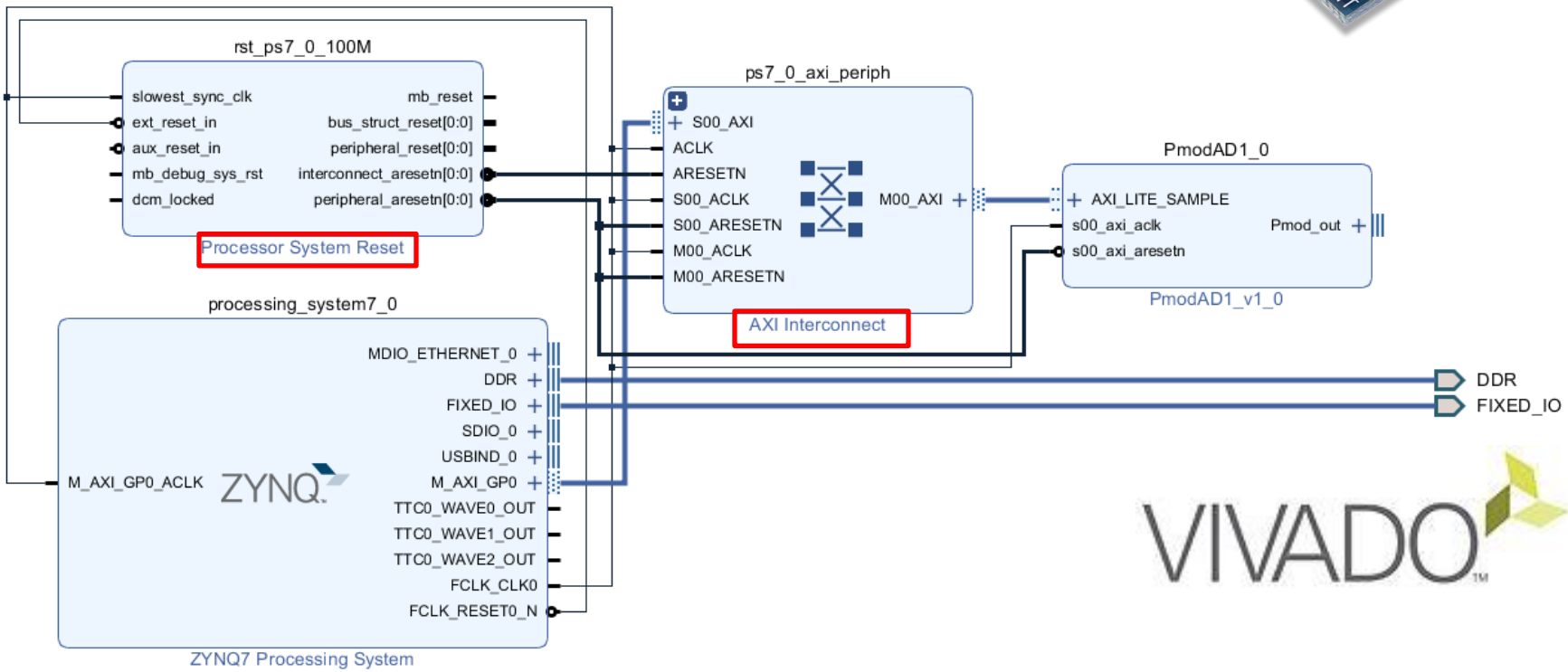




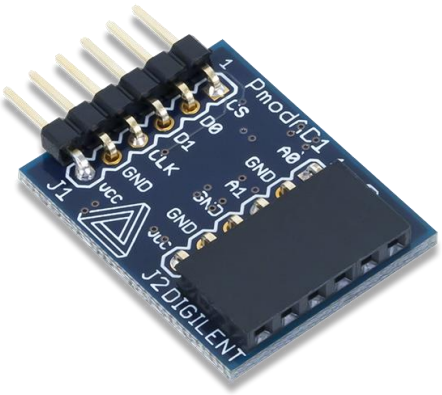
VIVADO™

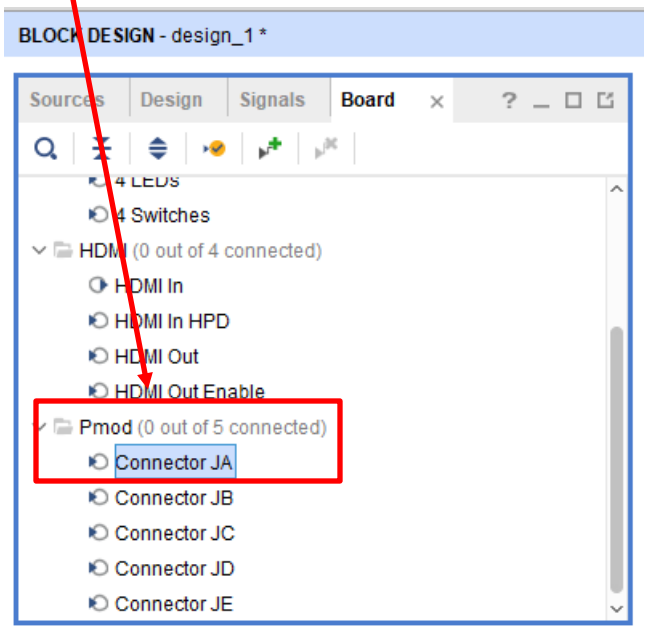
Zynq SPI Peripheral - ADC. Run *Connection Automation* to add the *Reset* and *AXI interconnect*.

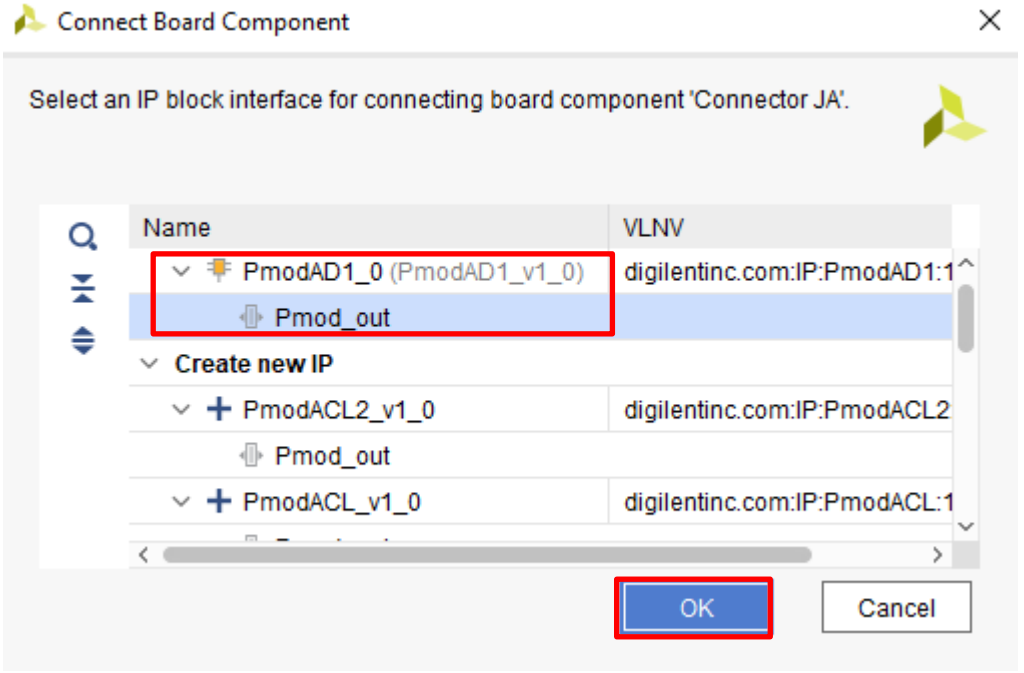




Zynq SPI Peripheral - ADC. In the *Block Design…Board* tab double-click *Pmod Connector JA* and connect *Pmod_out*. *OK* to continue.

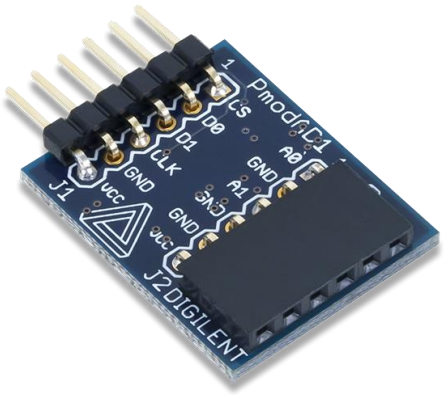


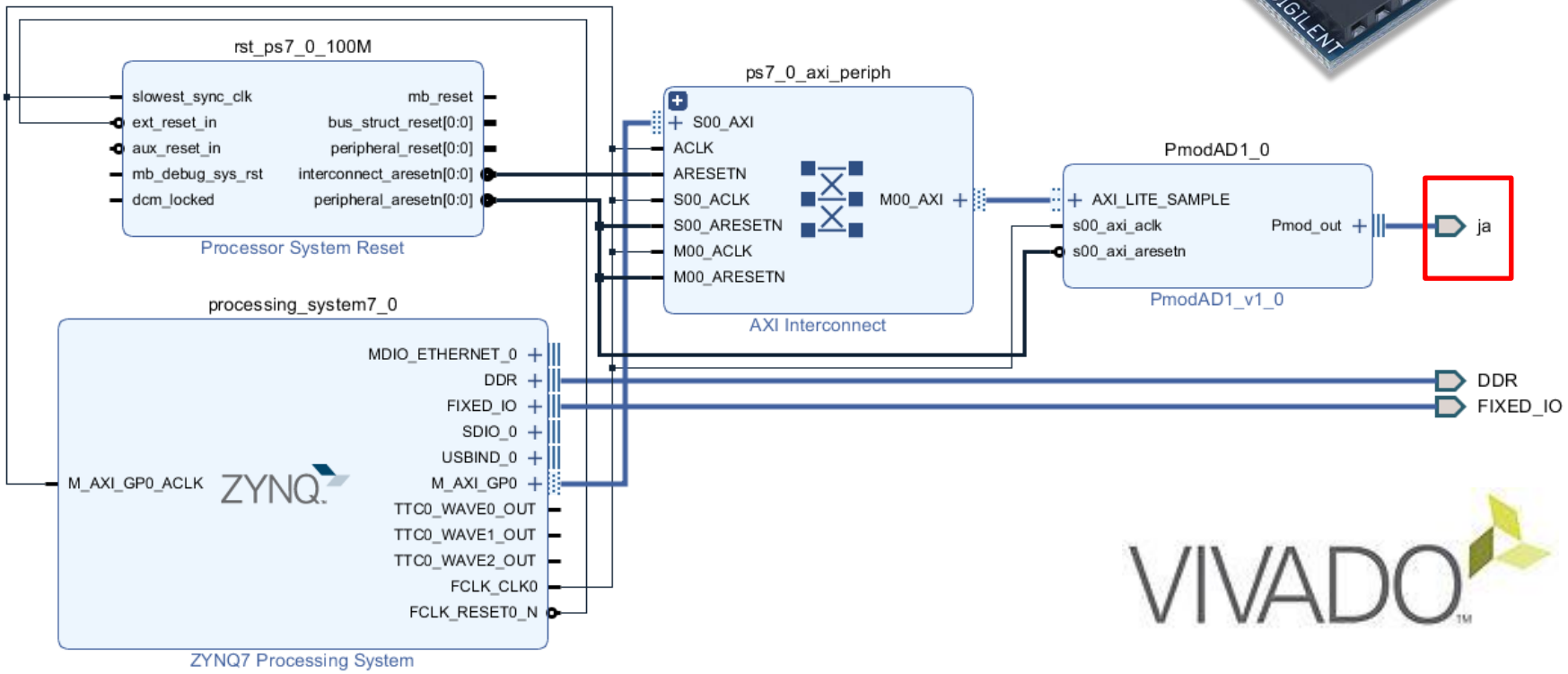




VIVADO™

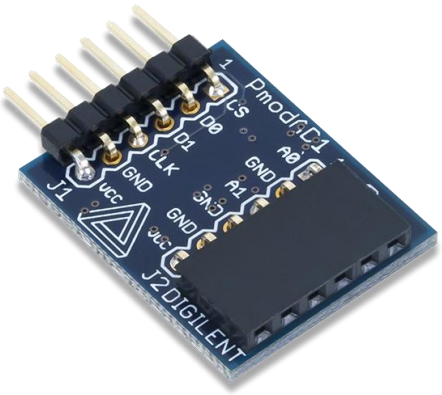
Zynq SPI Peripheral - ADC. The external Pmod port *JA* is added to the block design.

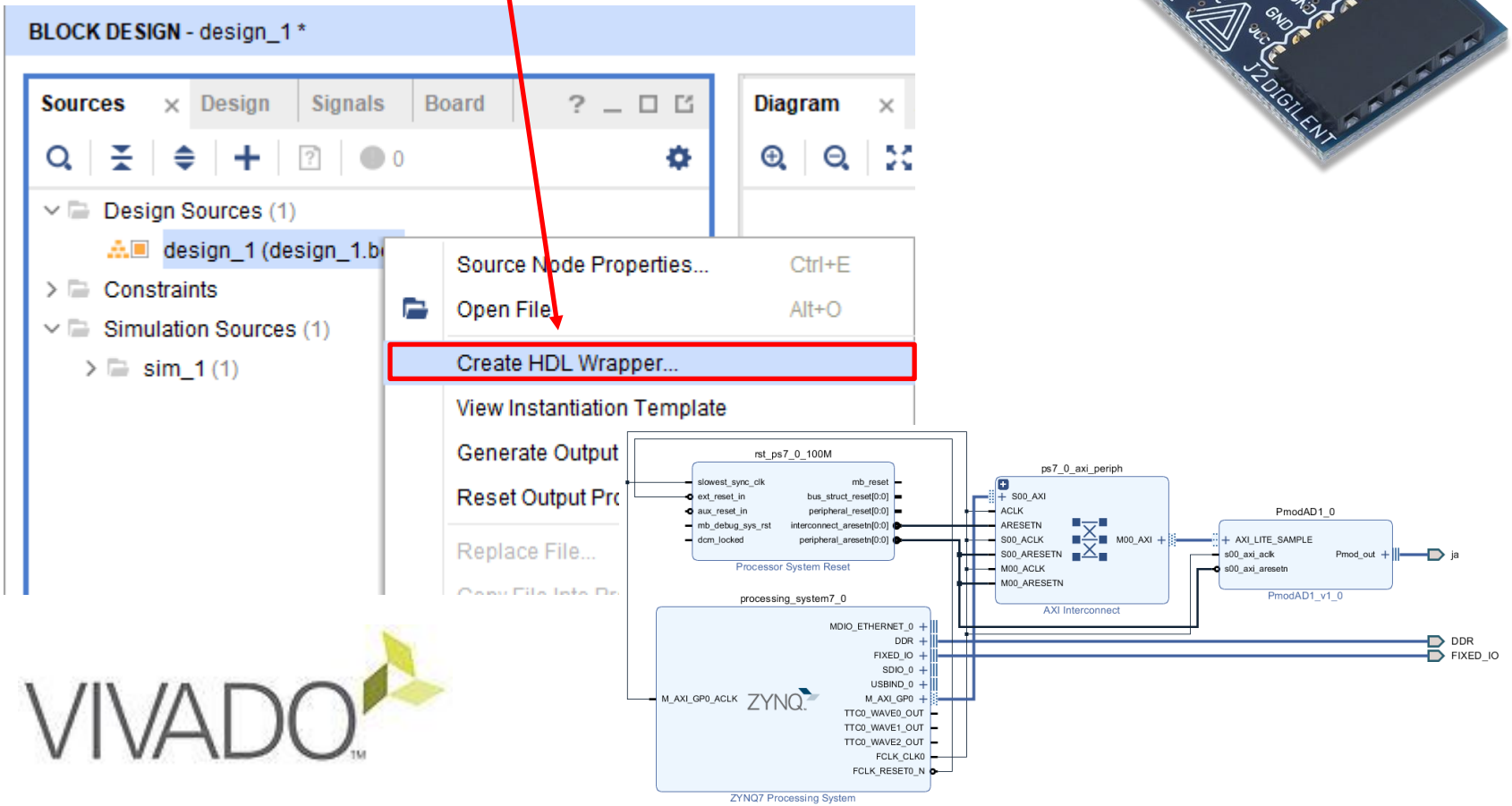




VIVADO™

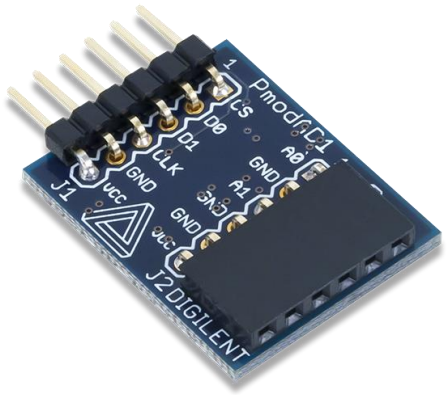
Zynq SPI Peripheral - ADC. In the *Block Design*…*Sources* tab right-click *design_1* and *Create HDL Wrapper*.



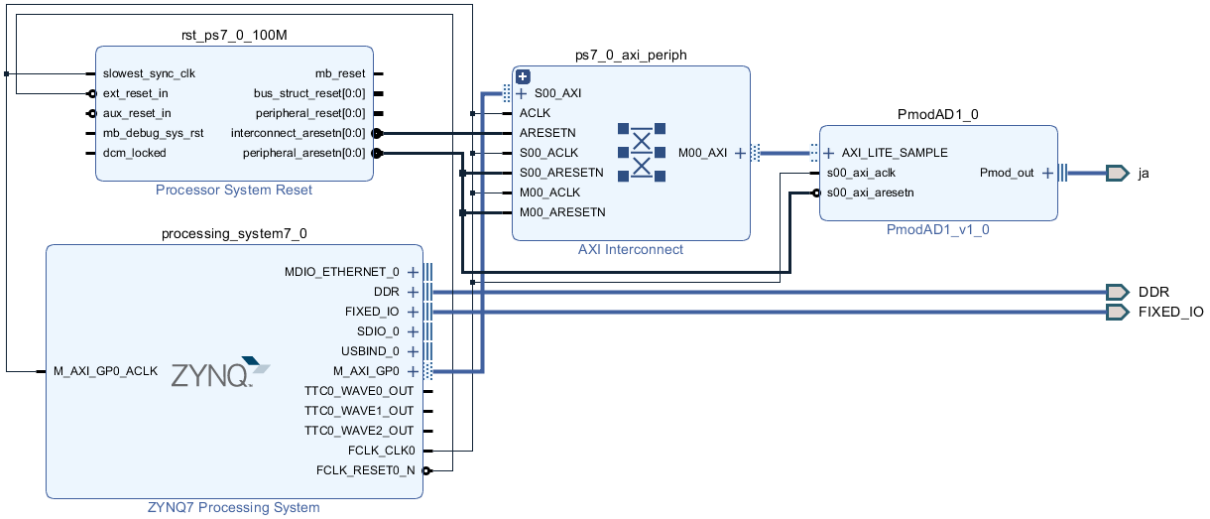


VIVADO™

Zynq SPI Peripheral - ADC. In the *Flow Navigator*…*Generate Bitstream* for the Hardware design. Select *File*…*Export*… *Hardware*. Remember to include bitstream ✓ then *File*…*Launch SDK*.

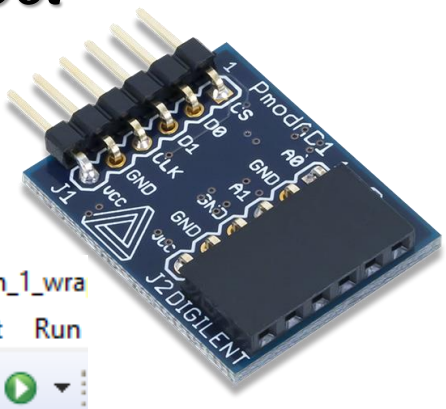


SDK
Software Development Kit



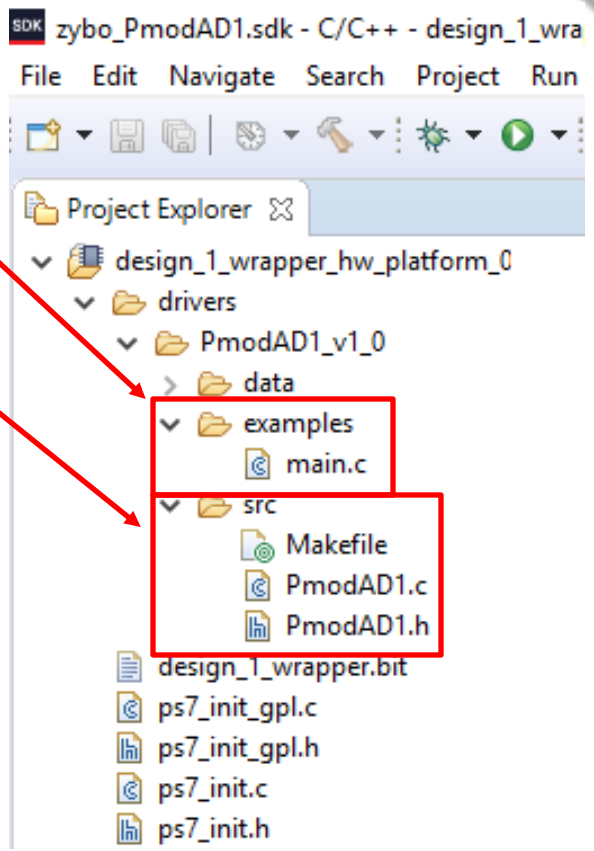
VIVADO™

Zynq SPI Peripheral - ADC. In *SDK…Project Explorer…Drivers* the PmodAD1 software should appear.

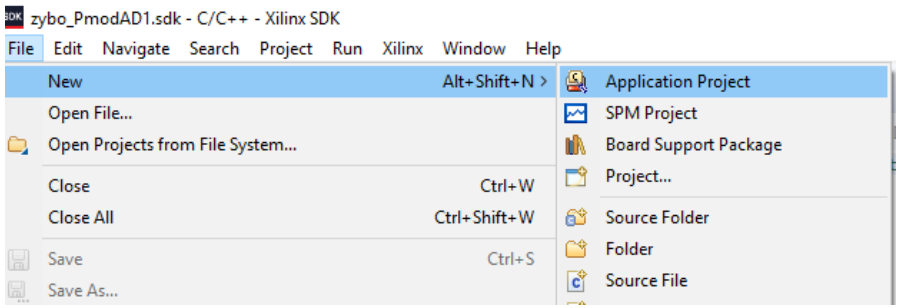


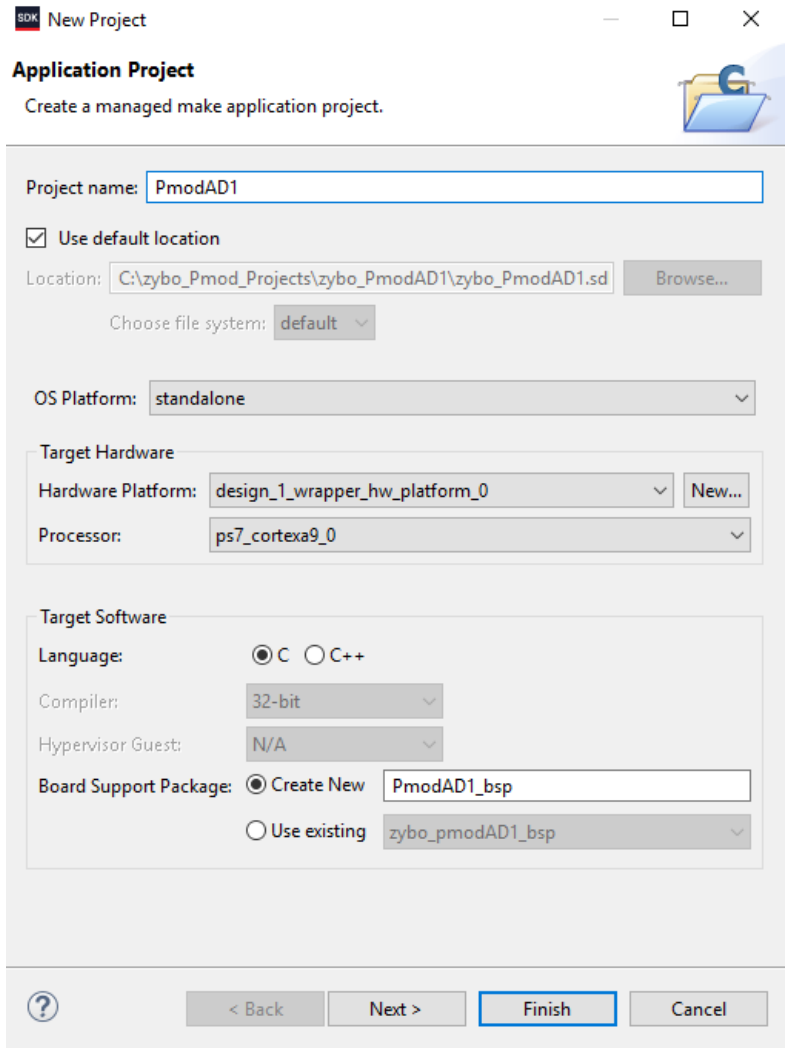
Under examples *main.c* is the project demonstration file.

Under *src PmodAD1.c* is the driver source code and *PmodAD1.h* is the include file for definitions.

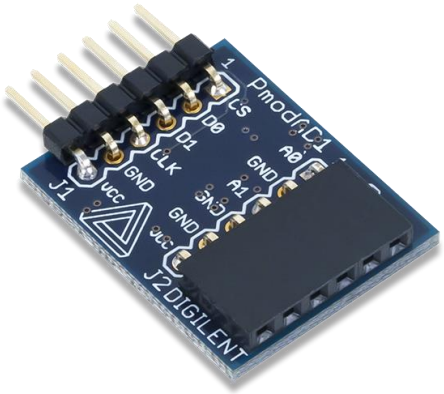


Zynq SPI Peripheral - ADC. In *SDK* open *File…New…Application Project*





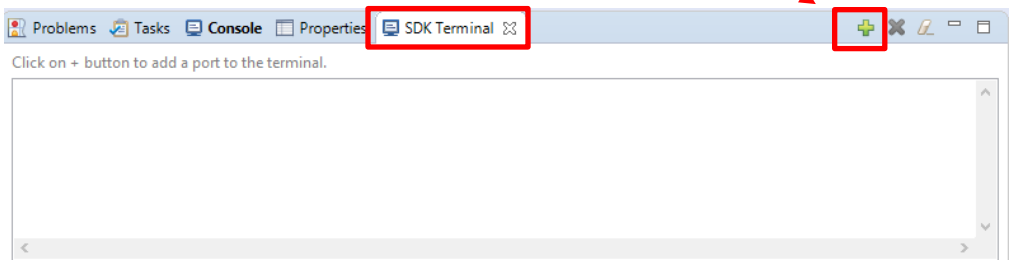
Project name is *PmodAD1* (only for SDK) and all other options as provided.

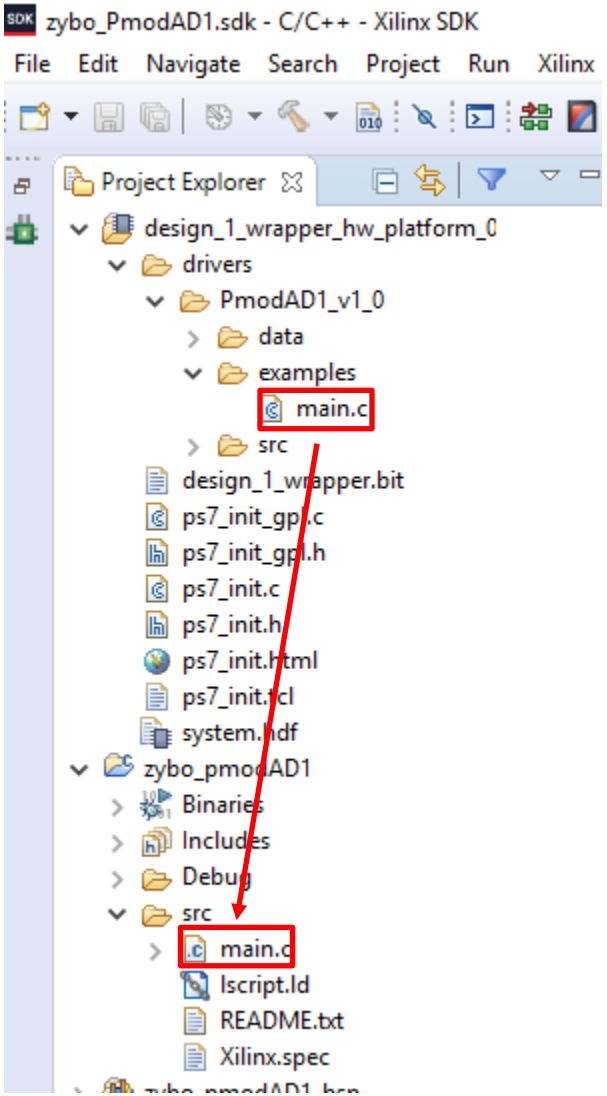


Zynq SPI Peripheral - ADC. In *SDK…Project Explorer…Drivers* copy *main.c* and paste it to *zybo_pmodAD1…src*.

Delete the *helloworld.c* program from this folder.

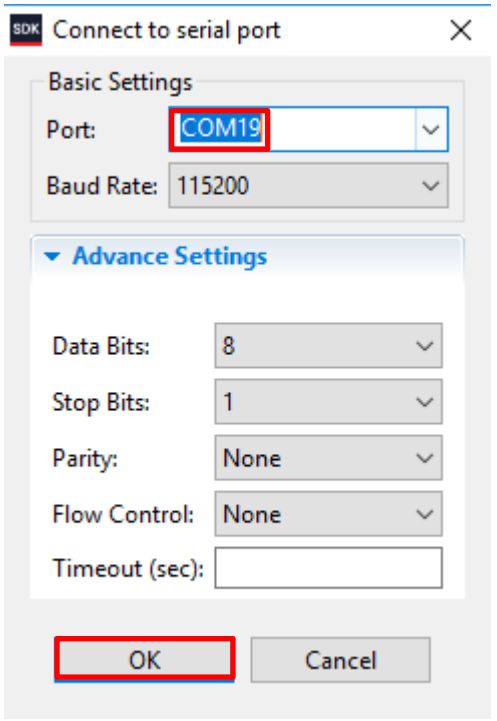
The program output appears in the SDK Terminal. Click + to open a terminal port.

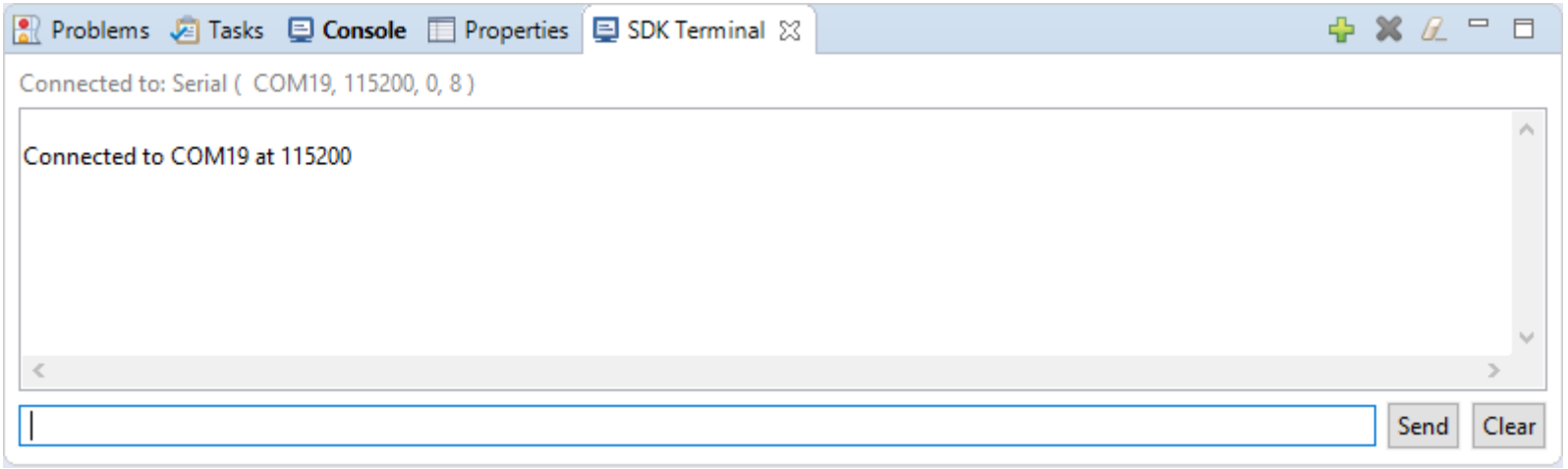




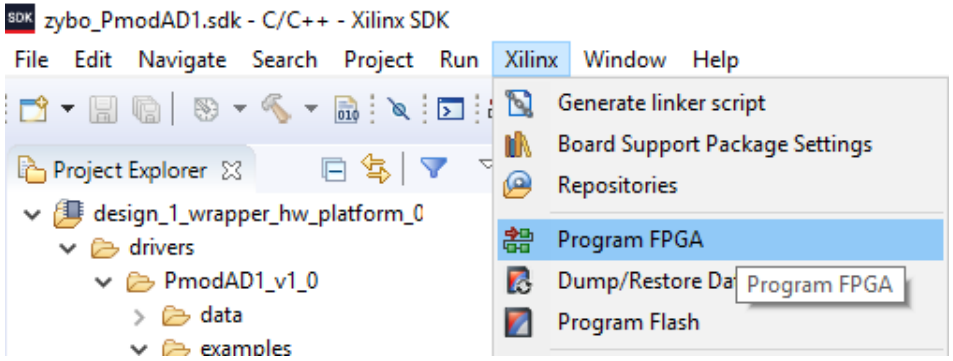
Zynq SPI Peripheral – ADC. The Port (may be) automatically identified. This is the same USB port used for programming the Zybo and providing power.

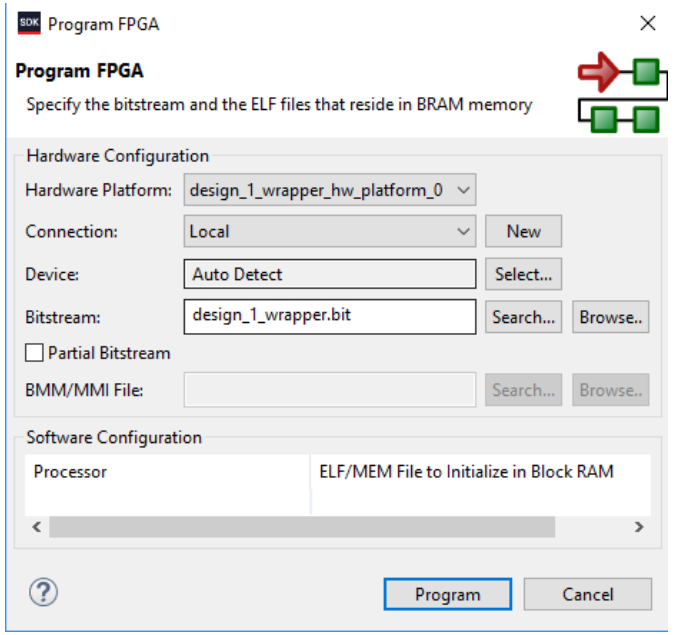
The default baud (bit) rate, data and stop bits, parity and flow control are used. *OK* to continue.

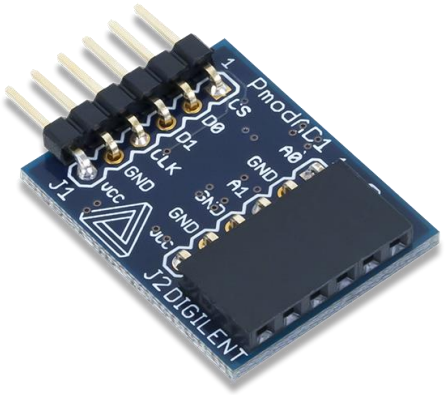




Zynq SPI Peripheral - ADC. In *SDK* on the *Xilinx* tab select *Program FPGA*.

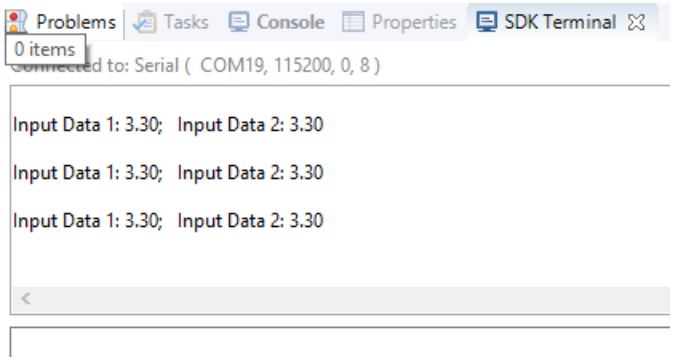


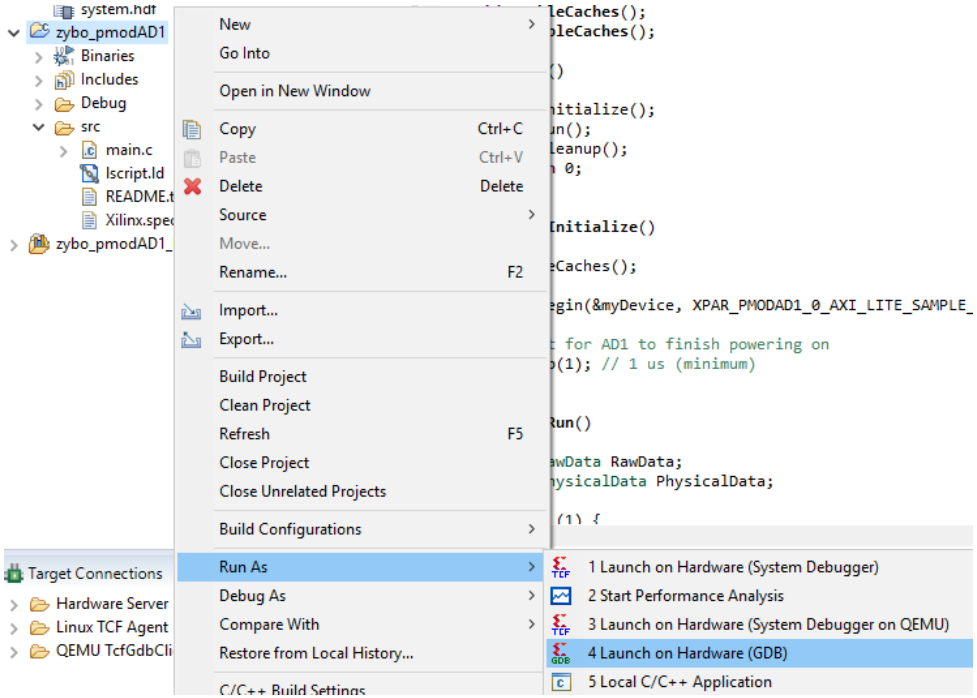




Zynq SPI Peripheral - ADC. In *SDK…Project Explorer* under the project name *zybo_PmodAD1* right-click and select *Run As…Launch on Hardware (GDB).*

Program output appears in the SDK terminal. Source code changes automatically saved and re-built before execution.

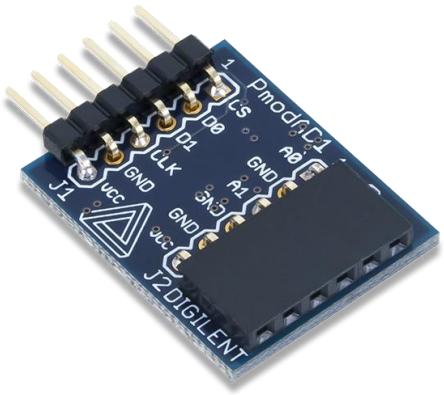




Zynq SPI Peripheral - ADC. *main.c*

system.hdf | PmodAD1.c | main.c | PmodAD1.h

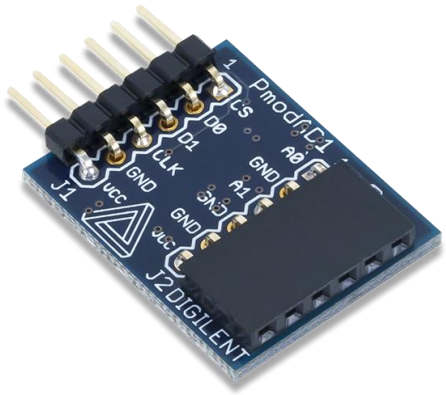
```
/*     main.c --     PmodAD1 Example Project                    */

#include <stdio.h>
#include "xparameters.h"
#include "xil_io.h"
#include "xil_types.h"
#include "xil_cache.h"
#include "PmodAD1.h"
#include "sleep.h"

PmodAD1 myDevice;
const float ReferenceVoltage = 3.3;

void DemoInitialize();
void DemoRun();
void DemoCleanup();
void EnableCaches();
void DisableCaches();

int main ()
{
    DemoInitialize();
    DemoRun();
    DemoCleanup();
    return 0;
}

void DemoInitialize()
{
    EnableCaches();

    AD1_begin(&myDevice, XPAR_PMODAD1_0_AXI_LITE_SAMPLE_BASEADDR);

    //wait for AD1 to finish powering on
    usleep(1); // 1 us (minimum)
}
```



Zynq SPI Peripheral - ADC. *main.c*

system.hdf | PmodAD1.c | main.c | PmodAD1.h

```
void DemoRun()
{
    AD1_RawData RawData;
    AD1_PhysicalData PhysicalData;

    while (1) {
        AD1_GetSample (&myDevice, &RawData); // capture raw samples
        AD1_RawToPhysical(ReferenceVoltage, RawData, &PhysicalData);
                    // convert raw samples into floats scaled to 0 - VDD

        printf("Input Data 1: %.02f;   ", PhysicalData[0]);
        printf("Input Data 2: %.02f\r\n", PhysicalData[1]);

        // do this 10x per second
        usleep(100000);
    }
}

void DemoCleanup()
{
    DisableCaches();
}
```



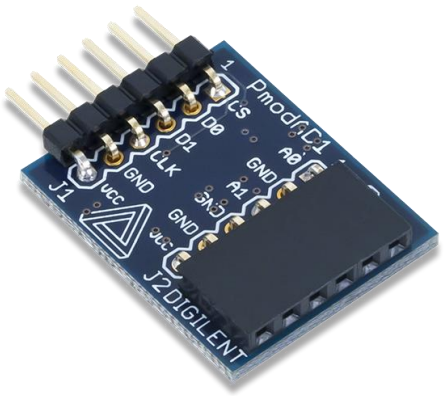
Zynq SPI Peripherals – AD1. *main.c* is also on Canvas as a convenient reference as *main.c.PmodAD1.pdf*.

main.c.PmodAD1.pdf - Adobe Reader

File Edit View Window Help

Open 1 / 2 135%

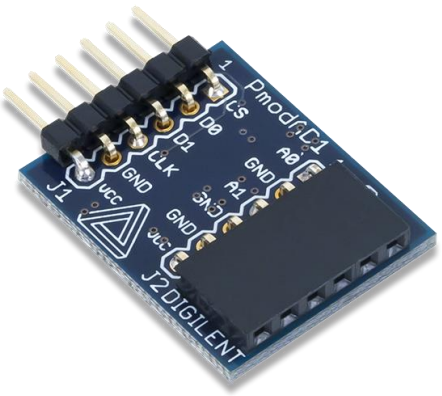
```
// main.c for PmodAD1 Vivado project
// ECE 3622

#include <stdio.h>
#include "xparameters.h"
#include "xil_io.h"
#include "xil_types.h"
#include "xil_cache.h"
#include "PmodAD1.h"
#include "sleep.h"

PmodAD1 myDevice;
const float ReferenceVoltage = 3.3;

void DemoInitialize();
void DemoRun();
void DemoCleanup();
void EnableCaches();
void DisableCaches();

int main ()
{
      DemoInitialize();
      DemoRun();
      DemoCleanup();
      return 0;
}
```



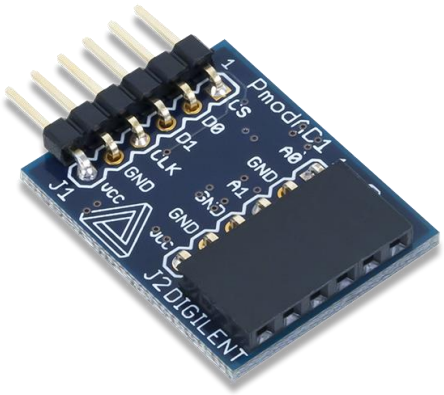
Zynq SPI Peripheral - ADC. *PmodAD1.c*

system.hdf | *PmodAD1.c | main.c | PmodAD1.h

```
/*      PmodAD1.c --      PmodAD1 Driver Source                                */

/***************************** Include Files *******************************/

#include "PmodAD1.h"

/************************** Function Definitions ***************************/

/* ------------------------------------------------------------ */
/***    void AD1_begin(PmodAD1* InstancePtr, u32 BaseAddress)
**
**  Parameters:
**      InstancePtr: A PmodAD1 object to start
**      BaseAddress: The base address of the PmodAD1 AXI_LITE_SAMPLE interface
**
**  Description:
**      Initialize the PmodAD1 device - note that the AD1 IP is free-running, and this
**      function just prepares the driver for use.
*/
void AD1_begin(PmodAD1* InstancePtr, u32 BaseAddress)
{
    InstancePtr->BaseAddress=BaseAddress;
}
```



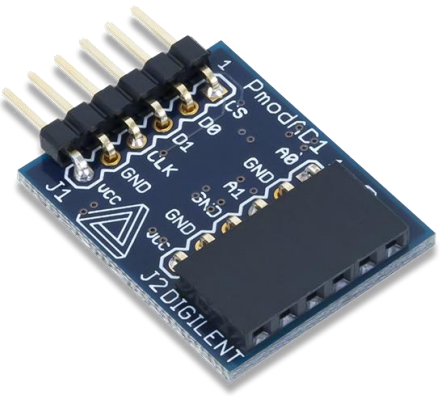
Zynq SPI Peripheral - ADC. *PmodAD1.c*




```
/* ------------------------------------------------------------ */
/***    void AD1_GetSample(PmodAD1* InstancePtr, AD1_RawData *RawDataPtr)
**
**  Parameters:
**      InstancePtr:    A PmodAD1 object to start
**      RawDataPtr:     Pointer to an array of raw data to return values in
**
**  Return:
**      *RawDataPtr: an array of unsigned 16 bit integers to store the ADC data in.
**          Channel 1's 12 bit data is stored in the RawData array at position 0
**          Channel 2's 12 bit data is stored in the RawData array at position 1
**
**  Description:
**      This function captures the most recently read sample from the Pmod AD1 IP core.
*/
void AD1_GetSample(PmodAD1* InstancePtr, AD1_RawData *RawDataPtr)
{
    u32 data;
    data = Xil_In32(InstancePtr->BaseAddress);
    (*RawDataPtr)[0] = data & AD1_DATA_MASK;
    (*RawDataPtr)[1] = (data >> 16) & AD1_DATA_MASK;
}

/*                                                              */
```

Zynq SPI Peripheral - ADC. *PmodAD1.c*

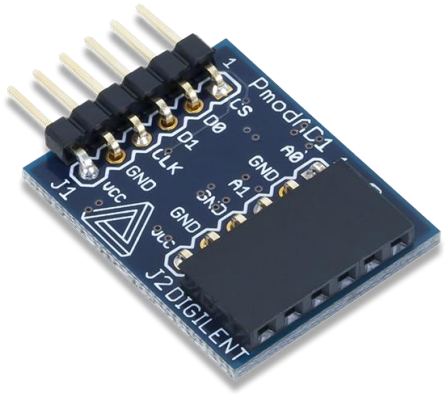
```
/* ------------------------------------------------------------ */
/***    void AD1_RawToPhysical(float ReferenceVoltage, AD1_RawData RawData, AD1_PhysicalData *PhysicalDataPtr)
**
**  Parameters:
**      ReferenceVoltage:   floating point representation of the voltage available to the AD1's VCC pin.
**      RawData:            array of raw 12 bit sample data received from the Pmod AD1
**      PhysicalDataPtr:    Pointer to an array of floats to return values in
**
**  Return:
**      *PhysicalDataPtr:   an array of floats to store the converted data in.
**
**  Description:
**      This function converts an AD1 sample to a human readable value.
*/
void AD1_RawToPhysical(float ReferenceVoltage, AD1_RawData RawData, AD1_PhysicalData *PhysicalDataPtr)
{
    (*PhysicalDataPtr)[0] = ((float)RawData[0]) * (ReferenceVoltage / ((1 << AD1_NUM_BITS) - 1));
    (*PhysicalDataPtr)[1] = ((float)RawData[1]) * (ReferenceVoltage / ((1 << AD1_NUM_BITS) - 1));
}
```



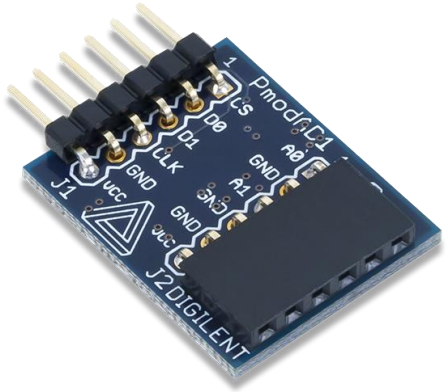
Zynq SPI Peripheral - ADC. *PmodAD1.h*

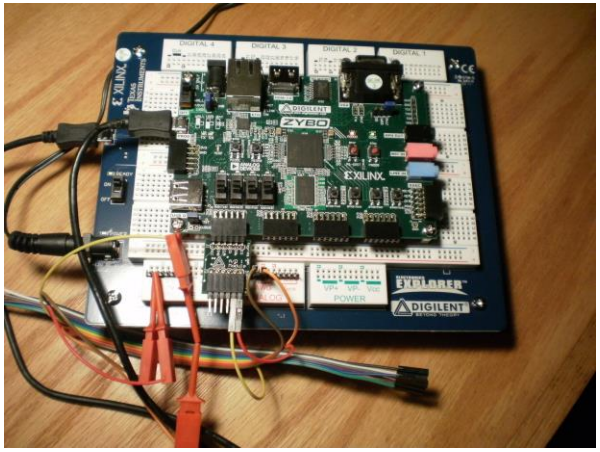
system.hdf | PmodAD1.c | main.c | *PmodAD1.h

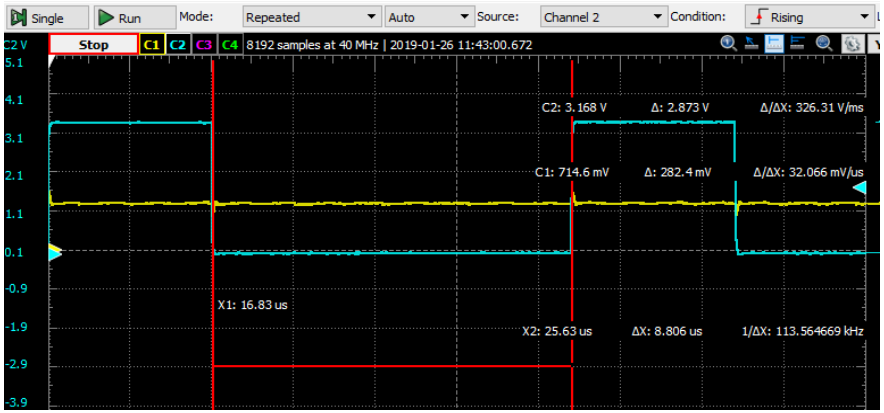
```
/*     PmodAD1.h --     PmodAD1 Driver Definitions                    */

#ifndef PMODAD1_H
#define PMODAD1_H

/****************** Include Files ********************/

#include "xil_types.h"
#include "xil_io.h"

/* ------------------------------------------------------------ */
/*                  Definitions                                 */
/* ------------------------------------------------------------ */

#define AD1_NUM_BITS 12
#define AD1_DATA_MASK 0xFFF

typedef struct PmodAD1 {
    u32 BaseAddress;
} PmodAD1;

typedef u16 AD1_RawData[2];
typedef float AD1_PhysicalData[2];

/* ------------------------------------------------------------ */
/*                  Procedure Declarations                      */
/* ------------------------------------------------------------ */

void AD1_begin(PmodAD1* InstancePtr, u32 SPI_Address);
void AD1_GetSample(PmodAD1* InstancePtr, AD1_RawData *RawDataPtr);
void AD1_RawToPhysical(float ReferenceVoltage, AD1_RawData RawData, AD1_PhysicalData *PhysicalDataPtr);

#endif // PMODAD1_H
```



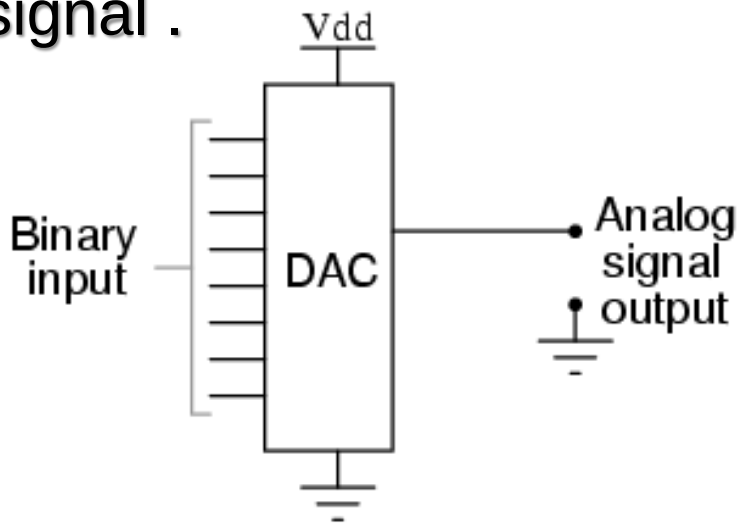
Zynq SPI Peripheral - ADC. For the PmodAD1 template project, the measured CS pulse is active low for 8.8 µsec and inactive high for 4.83 µsec. The active duty cycle for ADC conversion is 8.8/(4.83+8.8) ≈ 64.6%
and the data rate is
1/(13.63 µsec) ≈ 73.4 ksamples/sec

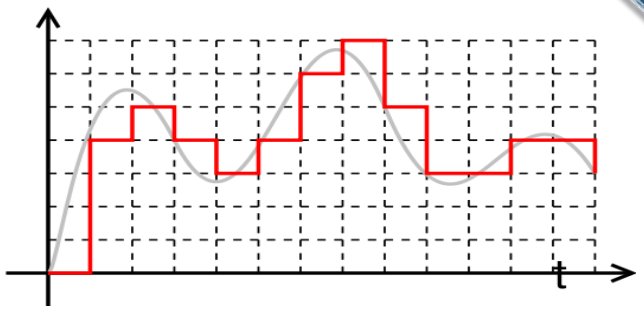


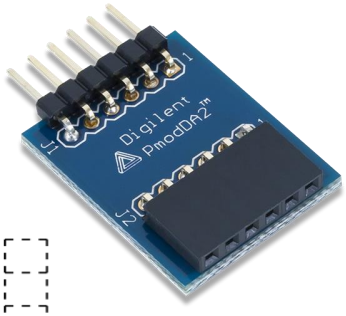


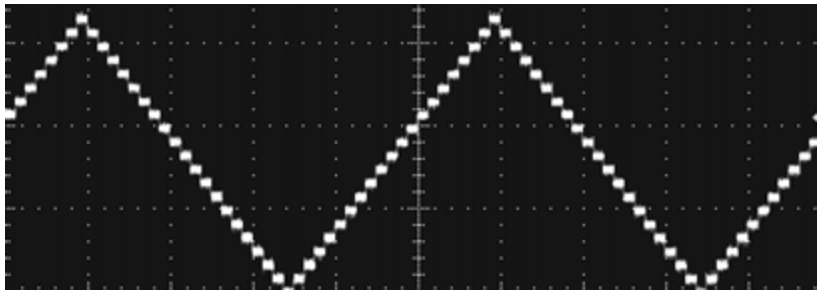


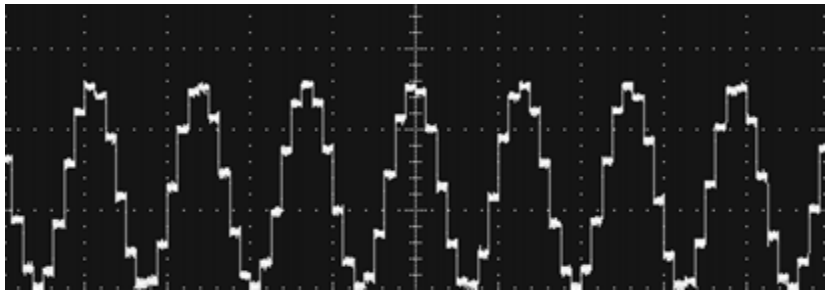
Zynq SPI Peripheral - DAC. The DAC provides an analog output signal for the discrete sampled data of an application as a *step-wise* but continuous analog output signal .







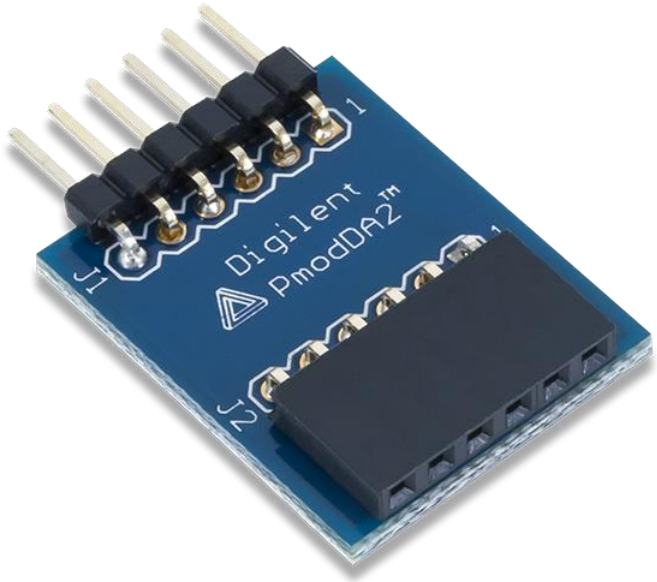




Zynq SPI Peripheral - DAC. The DAC provides a analog output signal for embedded system design in audio processing, analog and digital baseband and bandpass communication and digital process control.

The analog output voltage resolution (or step size) ΔV for a DAC that inputs an unsigned n-bit binary values is determined by:

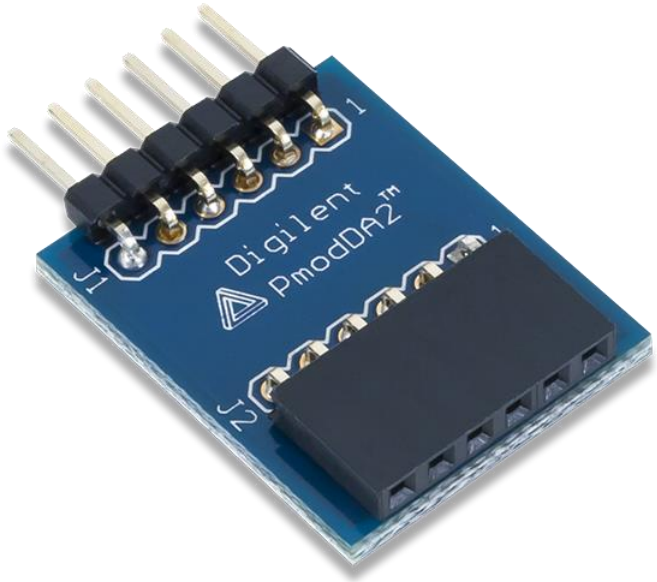
$$\Delta V = \frac{V_{REF}}{2^n}$$



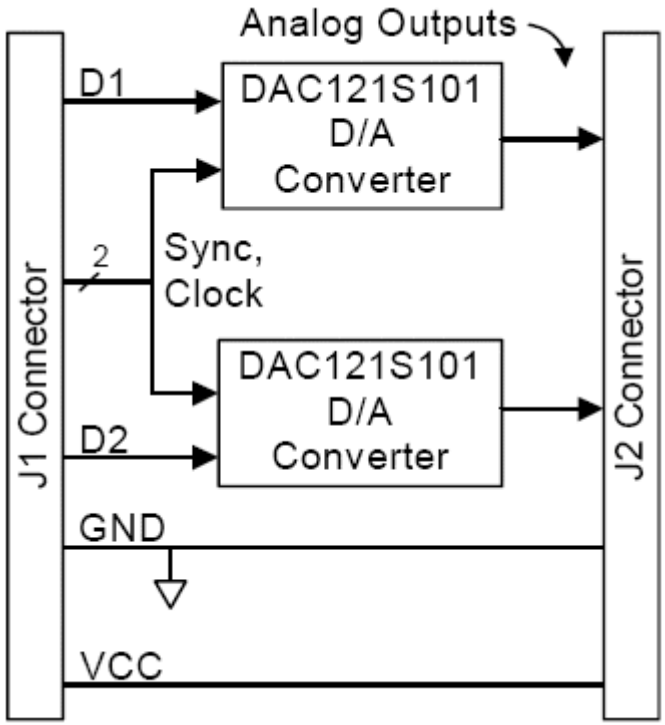
Zynq SPI Peripheral - DAC. V_{REF} is the precision DC voltage used as a reference. The maximum rate at which the DAC can accept binary data and update its analog voltage output is its throughput rate R_{DAC} in samples per second.

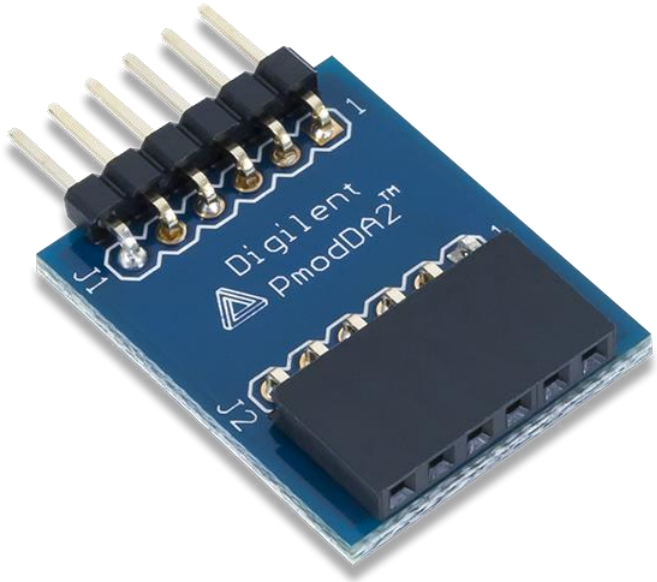
The Digilent PmodDA2 HAS two channels, each with 12-bits of resolution.

$$\Delta V = \frac{V_{REF}}{2^n}$$



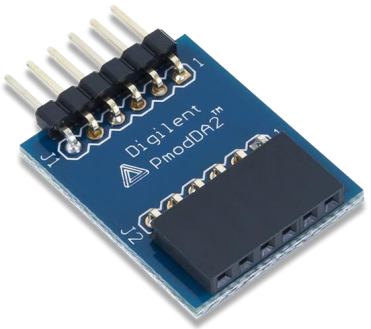
Zynq SPI Peripheral - DAC. The PmodDA2 hardware module features two Texas Instruments (*www.ti.com*) DAC121S101 12-bit DACs and utilizes the SPI protocol.





Zynq SPI Peripheral - DAC. PmodDA2 data protocol

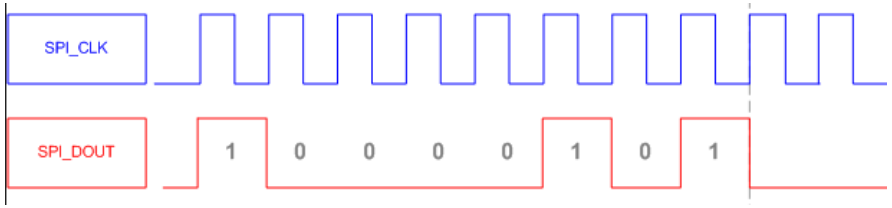
PmodDA2 National Semiconductor DAC121S101 DAC 16-bit data packet

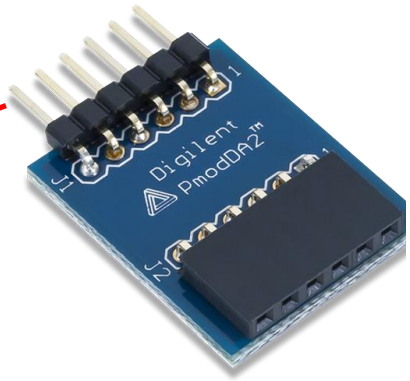


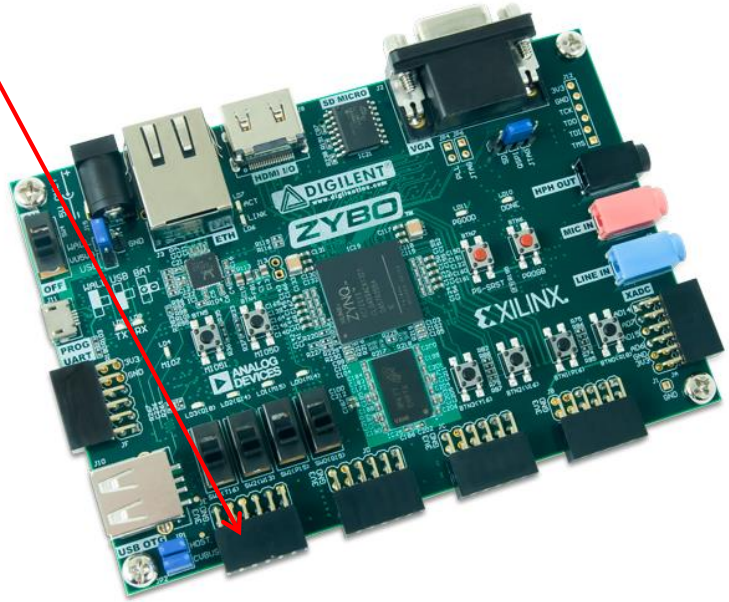
Bits	Contents (MSB…LSB)		
15-14	X (don't care)		
13-12	Command:	00	Normal operation
		01	Power-down with 1 kΩ to ground
		10	Power-down with 100 kΩ to ground
		11	Power-down with high impedance to ground
11-0	Data:	12-bit	unsigned DAC data, MSB first

Zynq SPI Peripheral - DAC. After all 16 bits have been sent, the sync signal SYNC is set to logic 1, whose rising edge starts the actual DAC process.

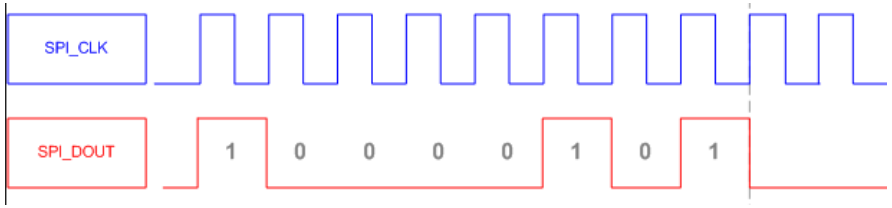
The PmodDA2 DAC is connected to the *bottom 6 pins* of the 12-pin peripheral hardware module connector (je) of the Zybo Board.

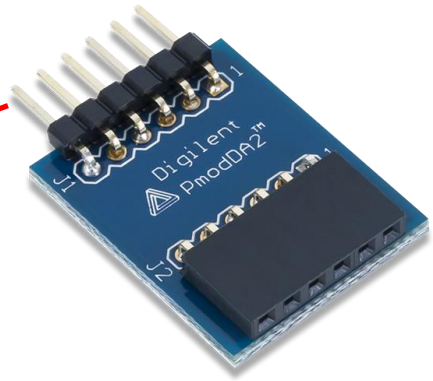


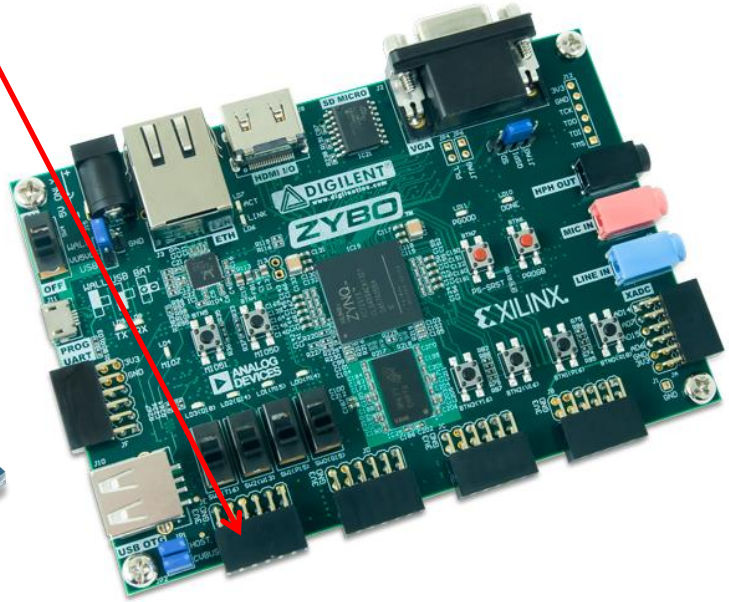




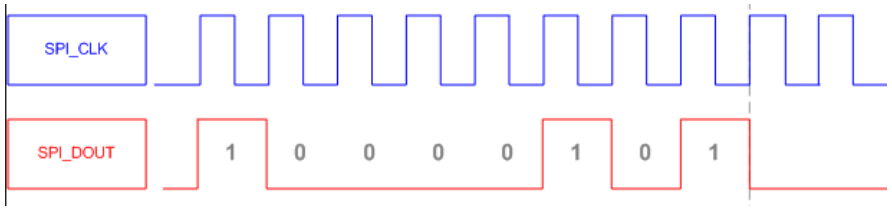
Zynq SPI Peripheral - DAC. The SYNC signal is pin je[7], the two input data signals DACD0 and DACD1 are pins je[8] and je[9] and the SPI bus clock signal SCLK is pin je[10]. Pins 11 and 12 are DC ground and power (VDD = +3.3 V DC).

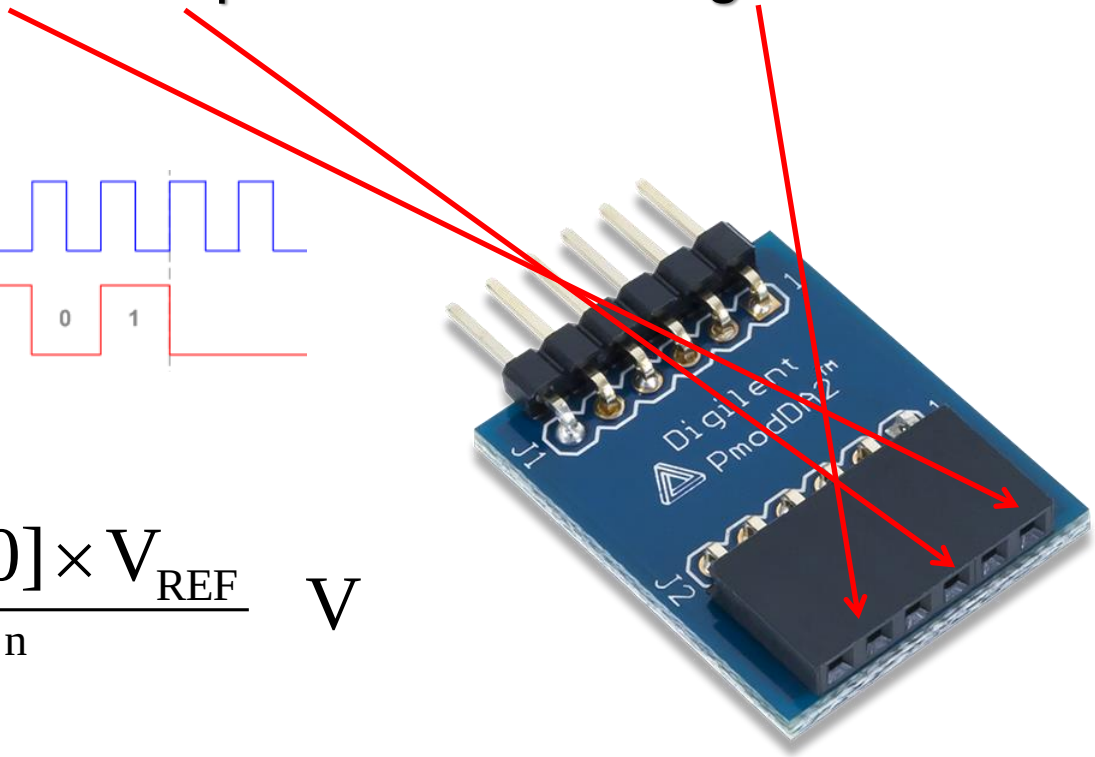






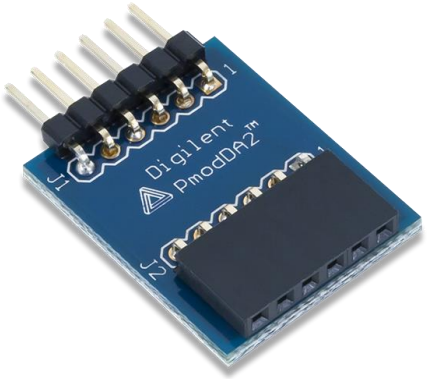
Zynq SPI Peripheral - DAC. The analog output voltage for the 12-bit DAC is determined by the equation below with n = 12, D[11:0] ranging from 0 to 4095 and V_{REF} = 3.3 V. The two unipolar analog output voltages A and B appear on a 6-pin header at pins 1 and 3 and ground at pin 5.

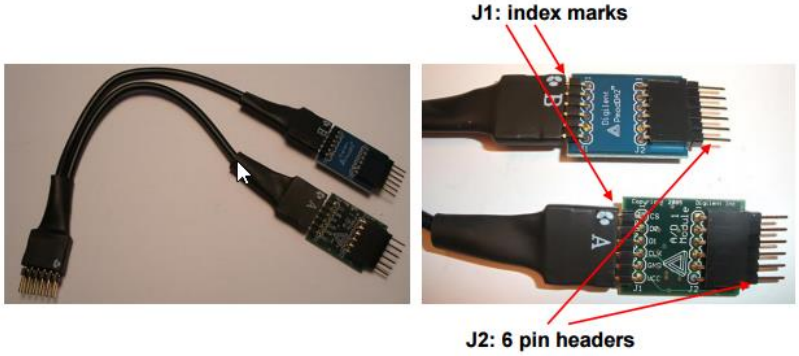




$$V_{OUT} = \frac{D[n-1:0] \times V_{REF}}{2^n} \quad V$$

Digital to Analog Converter. Installation and use of the DAC is described in the *PmodAD1 and PmodDA2 Peripherals* on *Canvas*.



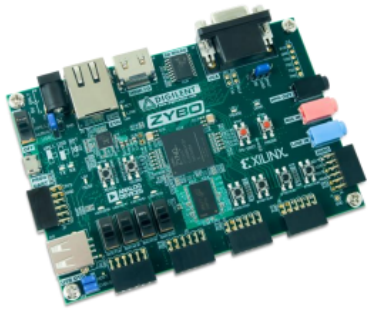


ECE3623 Embedded System Design Laboratory

Dennis Silage, PhD
silage@temple.edu

PmodAD1 and PmodDA2 Peripherals

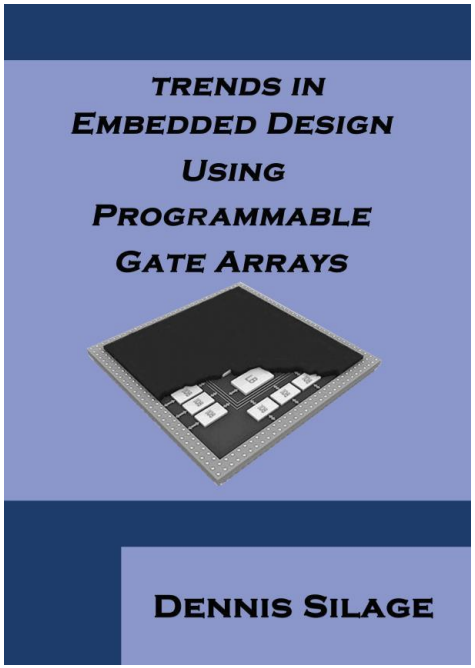
The Digilent Zybo Board does not feature an analog-to-digital converter (ADC) or digital-to-analog converter (DAC). The Digilent PmodAD1 and PmodDA2 provide this functionality and both 6-pin peripheral devices can be installed on the 12-pin Pmod jack on the lower left side of the Zybo Board.

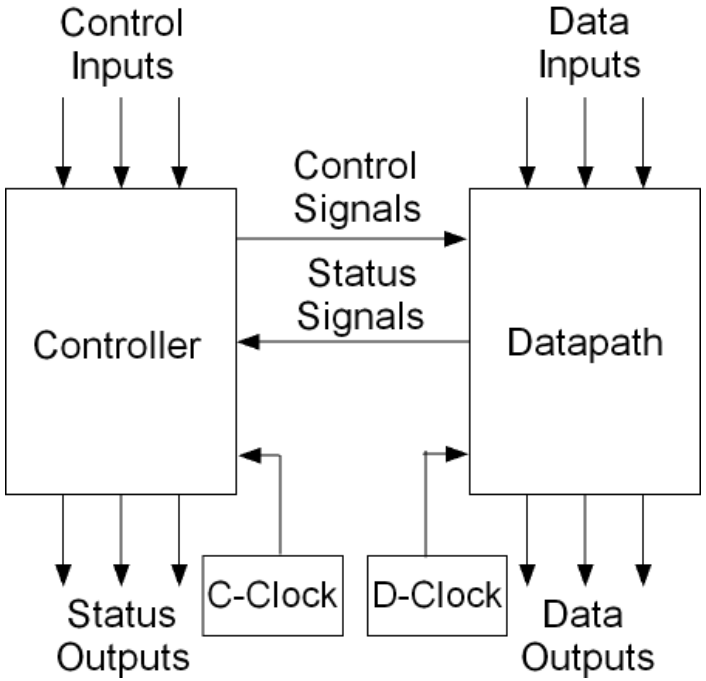


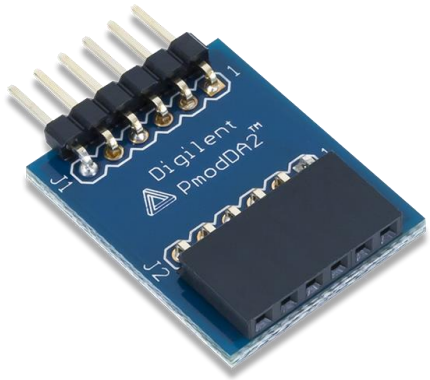
The leftmost, front Pmod jack (***je***) is available from the programmable logic (PL) of the Zynq SoC device and is shown as a *front view*. Note the location of pin 1 on the top right. Pin 7 is immediately below pin 1 on the bottom right. To facilitate the connection a 12 pin to two 6 pin cable is provided as shown in the Figure.

The PmodAD1 ADC is connected to the cable 6 pin connector A with the index mark positioned on pin 1 of J1 the SPI interface, as shown in the Pictures. The PmodDA2 is connected to the cable 6 pin connector B with the index mark position on pin 1 of J1 the SPI interface as shown in the Pictures.

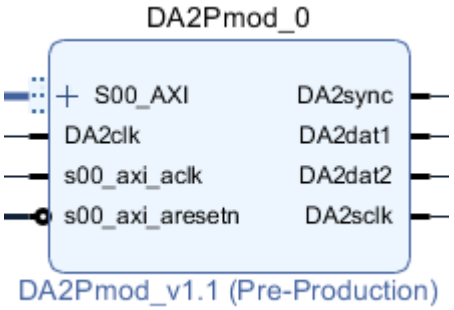
Zynq SPI Peripheral - DAC. The PmodDA2 DAC does not have an entry in the Digilent Vivado IP Library. However there is a controller-datapath Verilog model available for the PmodDA2.

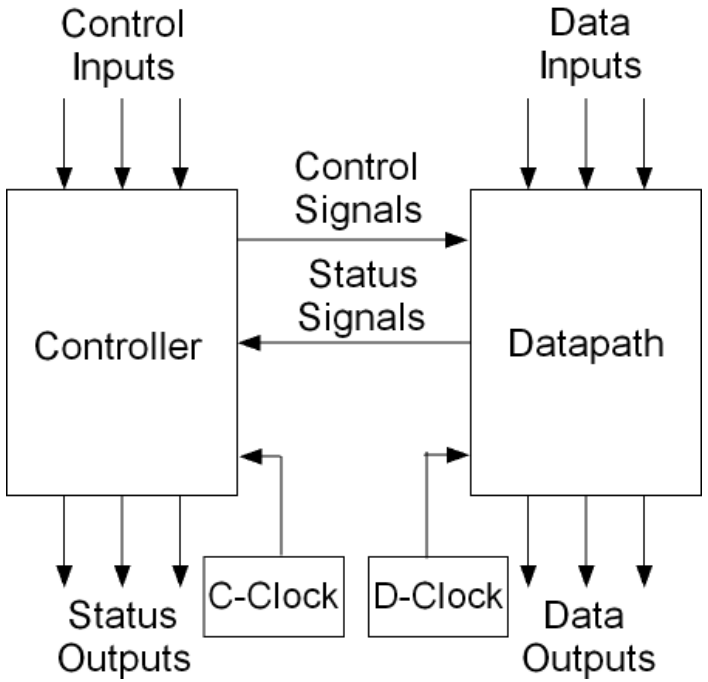


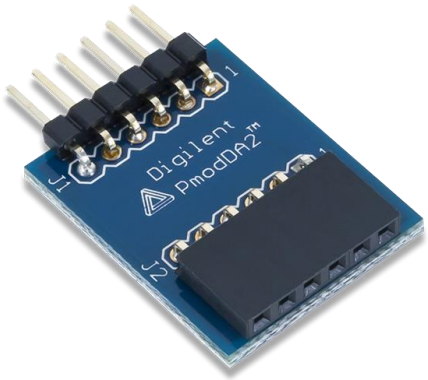




Zynq SPI Peripheral - DAC. The PmodDA2 DAC is implemented as a custom IP as *DA2Pmod_v1.1* using the original Verilog controller-datapath model. The control signal is *DA2acq* and the status signal is *DA2dav*.

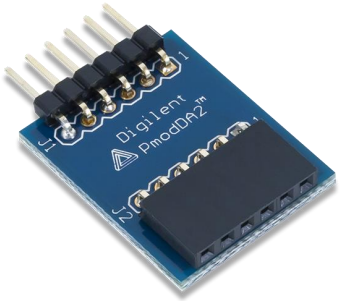






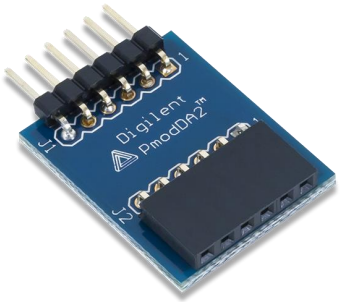
Zynq SPI Peripheral - DAC. As was demonstrated in Exercise 4A – Creating an IP in HDL, the instantiation file for the AXI Lite interface is *DA2Pmod_v1_0.v* and has the following initial entries for external ports:

// Users to add ports here	
output wire DA2sync,	SPI sync
output wire DA2dat1,	DA2 channel 1 serial data
output wire DA2dat2,	DA2 channel 2 serial data
output wire DA2sclk,	SPI sclk
input wire DA2clk,	state machine external clock
// User ports ends	
// Do not modify the ports beyond this line	



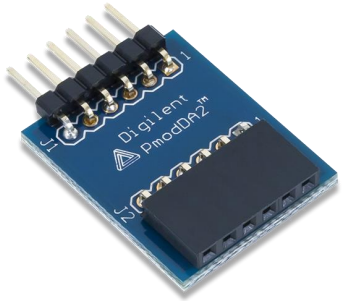
Zynq SPI Peripheral - DAC. The instantiation file for the AXI Lite interface is *DA2Pmod_v1_0.v* also has the following final entries for external ports:

```
DA2Pmod_v1_0_S00_AXI_inst (
        .DA2sync(DA2sync),
        .DA2dat1(DA2dat1),
        .DA2dat2(DA2dat2),
        .DA2sclk(DA2sclk),
        .DA2clk(DA2clk),
        .S_AXI_ACLK(s00_axi_aclk),
        .S_AXI_ARESETN(s00_axi_aresetn),
```



Zynq SPI Peripheral - DAC. The AXI Lite functionality file for is *DA2Pmod_v1_0_SOO_AXI.v* and has the following initial entries for external ports:

```
// Users to add ports here
output wire DA2sync,        SPI sync
output wire DA2dat1,        DA2 channel 1 serial data
output wire DA2dat2,        DA2 channel 2 serial data
output wire DA2sclk,        SPI sclk
input wire DA2clk,          state machine external clock
// User ports ends
// Do not modify the ports beyond this line
```



Zynq SPI Peripheral - DAC. The variable that indicates that the DA2Pmod has completed the output of data is:

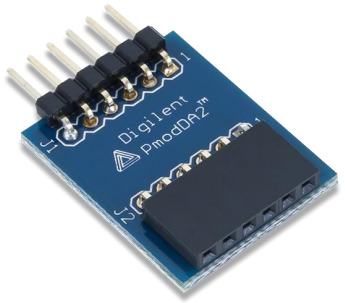
```
// DAC2Pmod address offset 0 - bit 0  DA2acq   DAC acquire output
//                          offset 4 - bit 0  DA2dav   DAC data available
//                          offset 8 -        DA2dat1  DAC channel 1 data
//                          offset 12 -       DA2dat2  DAC channel 2 data
```

which is read by the PS at address offset 4:

```
// Address decoding for reading registers
case (
axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
      2'h0   : reg_data_out <= slv_reg0;
      2'h1   : reg_data_out[0] <= DA2dav;
      2'h2   : reg_data_out <= slv_reg2;
      2'h3   : reg_data_out <= slv_reg3;
      default : reg_data_out <= 0;
endcase
```

DA2Pmod_v1_0_SOO_AXI.v

Zynq SPI Peripheral - DAC. The user logic starts with register definitions and assignments:

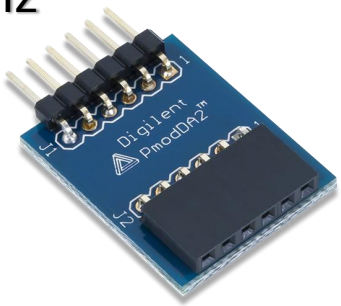



```
// Add user logic here
reg DA2syn=1;
reg DA2sck=1;
reg DA2dt1;
reg DA2dt2;
reg [C_S_AXI_DATA_WIDTH-1 : 0] DA2Areg;
reg [C_S_AXI_DATA_WIDTH-1 : 0] DA2Breg;
reg DA2acq;
reg DA2dav=0;

reg [5:0] dacstate=0;            //state register

assign DA2sync = DA2syn;     //external output signal definitions
assign DA2sclk = DA2sck;     //   assigned to internal registers
assign DA2dat1 = DA2dt1;
 assign DA2dat2 = DA2dt2;
```

DA2Pmod_v1_0_SOO_AXI.v

Zynq SPI Peripheral - DAC. Internal registers are read for the DA2Pmod data and the register DA2acq initiates the SPI data transfer. The AXI interface clock is a nominal 100 MHz.

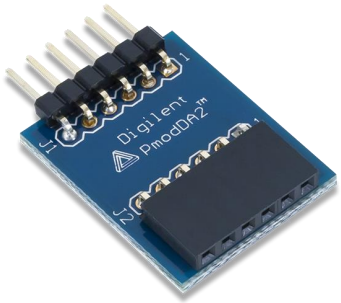
```
// DAC2Pmod address offset 0 - bit 0  DA2acq   DAC acquire output
//                         offset 4 - bit 0  DA2dav   DAC data available
//                         offset 8 -        DA2dat1  DAC channel 1 data
//                         offset 12 -       DA2dat2  DAC channel 2 data


always @( posedge S_AXI_ACLK)  //nominal 100 MHz
        begin
            DA2Areg <= slv_reg2;
            DA2Breg <= slv_reg3;
            DA2acq <= slv_reg0[0];
        end
```



DA2Pmod_v1_0_SOO_AXI.v

Zynq SPI Peripheral - DAC. The external DA2clk, which is provided by the Zynq PS, is 50 MHz. The SPI state machine transitions on every positive edge for a nominal 25 MHz clock which meets the specification of the PmodDA2.

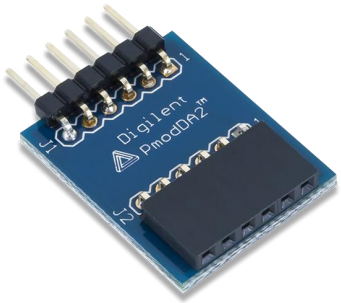
```
always @ ( posedge DA2clk)      //nominal 50 MHz, SPI clock 25 MHz
        begin
            if (DA2acq==0)       // DAC data?
                begin
                    dacstate=0;
                    DA2dav=0;    // DAC data NAK
                end
```



DA2Pmod_v1_0_SOO_AXI.v

Zynq SPI Peripheral - DAC. When the control signal and its register DA2acq is asserted and a completed SPI data transfer has occurred indicated by DA2dav, a new data transfer is initiated.

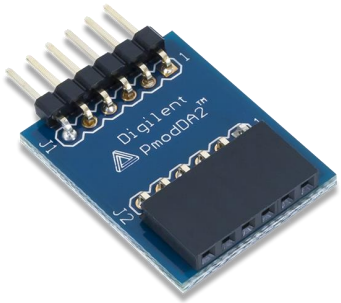
```
if (DA2acq==1 && DA2dav==0)    // DAC acqusition
   begin
        case (dacstate)
        0: begin
              DA2syn=1;
              DA2sck=1;
              dacstate=1;
           end
        1: begin
              DA2syn=0;
              DA2dt1=0;    // X don't care
              DA2dt2=0;    // X don't care
              dacstate=2;
```



DA2Pmod_v1_0_SOO_AXI.v

Zynq SPI Peripheral - DAC. There are 34 state transitions executing at 50 MHz for at least 680 nsec. This time does not include the AXI bus interface or the *main.c* overhead.

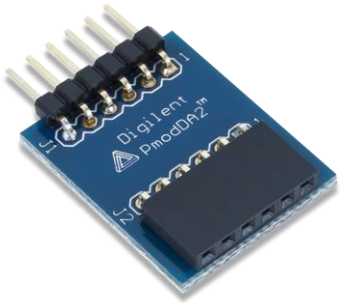
```
30: begin
      DA2sck=0;
      dacstate=31;
    end
31: begin
      DA2sck=1;
      DA2dt1=DA2Areg[0];
      DA2dt2=DA2Breg[0];
      dacstate=32;
    end
32: begin
      DA2sck=0;
      dacstate=33;
    end
```



DA2Pmod_v1_0_SOO_AXI.v

Zynq SPI Peripheral - DAC. The SPI state machine idles in state 34 until the control signal and its register DA2acq is de-asserted and then re-asserted to start the SPI data transfer again.

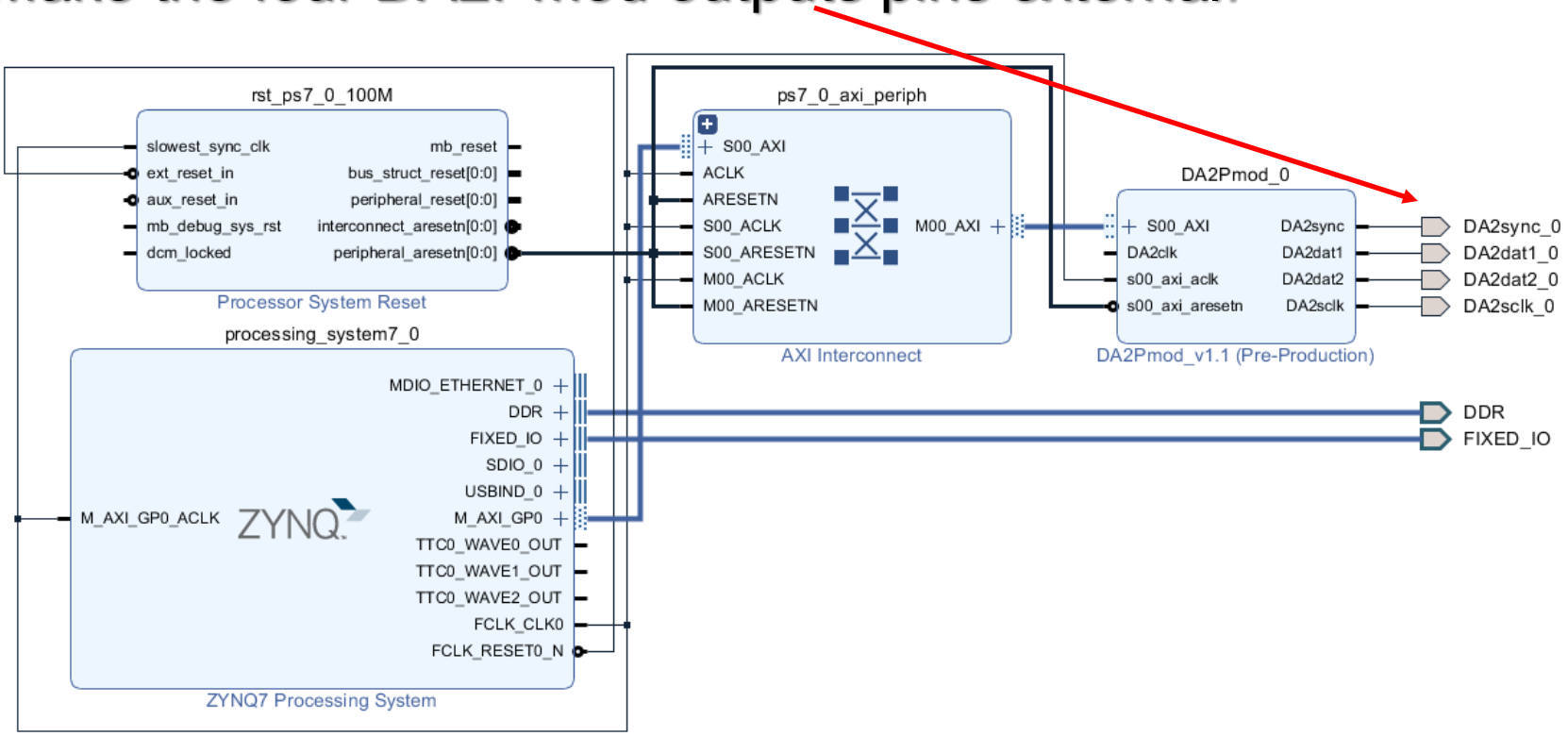
```
                33: begin
                      DA2syn=1;
                      DA2sck=1;
                      DA2dav=1;    // DAC data ACK
                      dacstate=34;
                    end
                34: dacstate=34;
                default: dacstate=34;
              endcase
            end
        end
endmodule
```



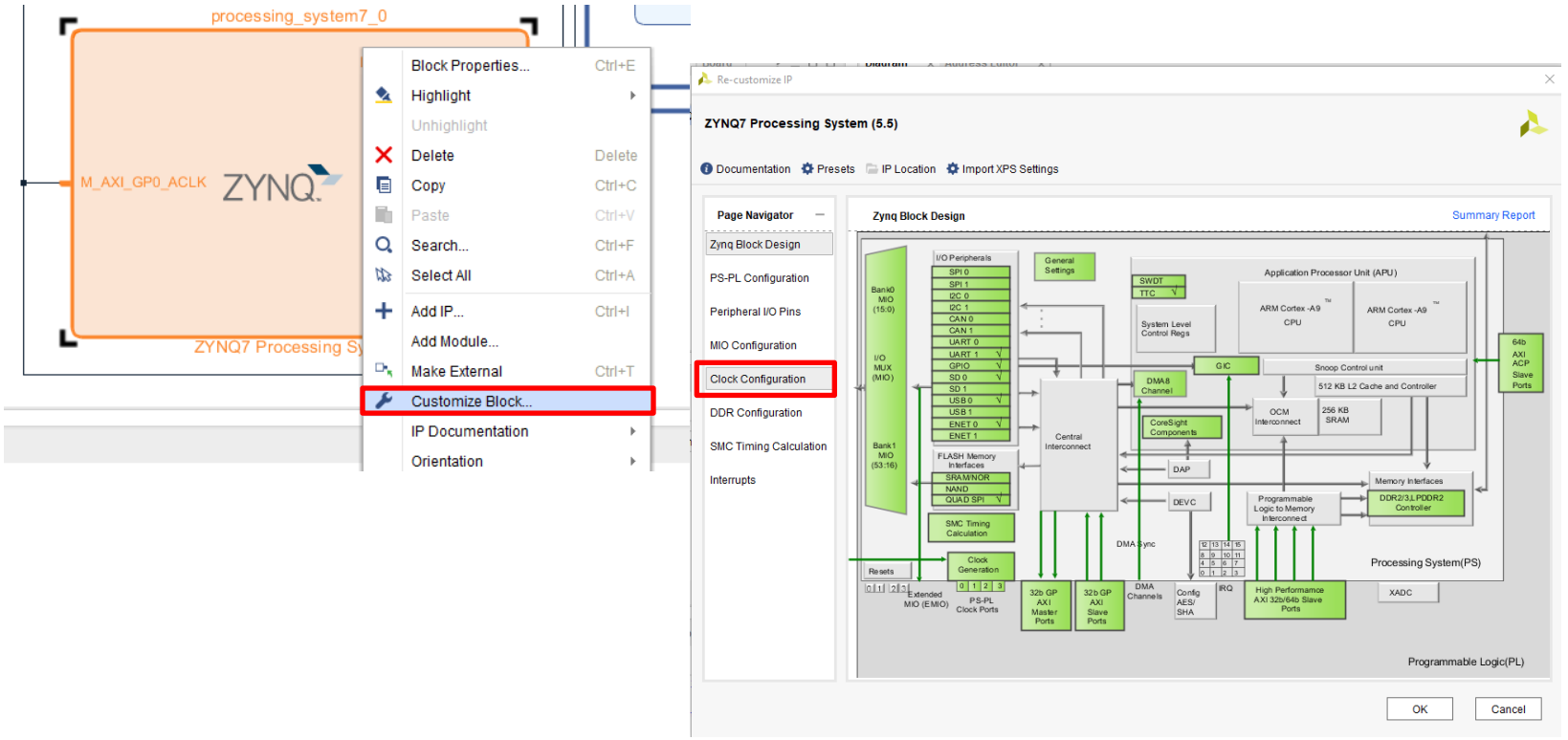
DA2Pmod_v1_0_SOO_AXI.v

Zynq SPI Peripheral - DAC. Copy the DA2Pmod IP to the *C:/ZynqPmodVivado-Library/ip* directory. Create a Vivado block design and add the DA2Pmod IP.

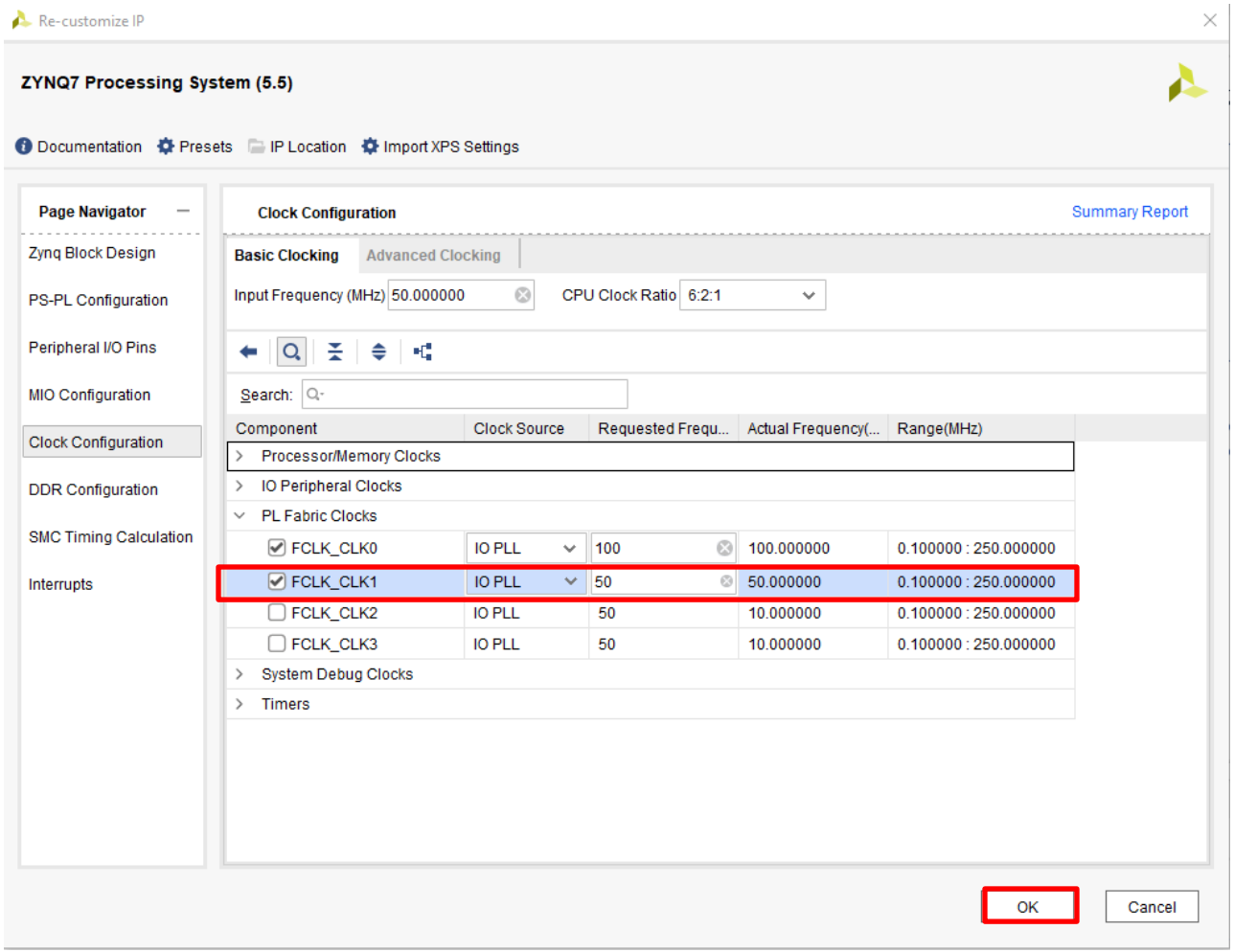
Make the four DA2Pmod outputs pins external.



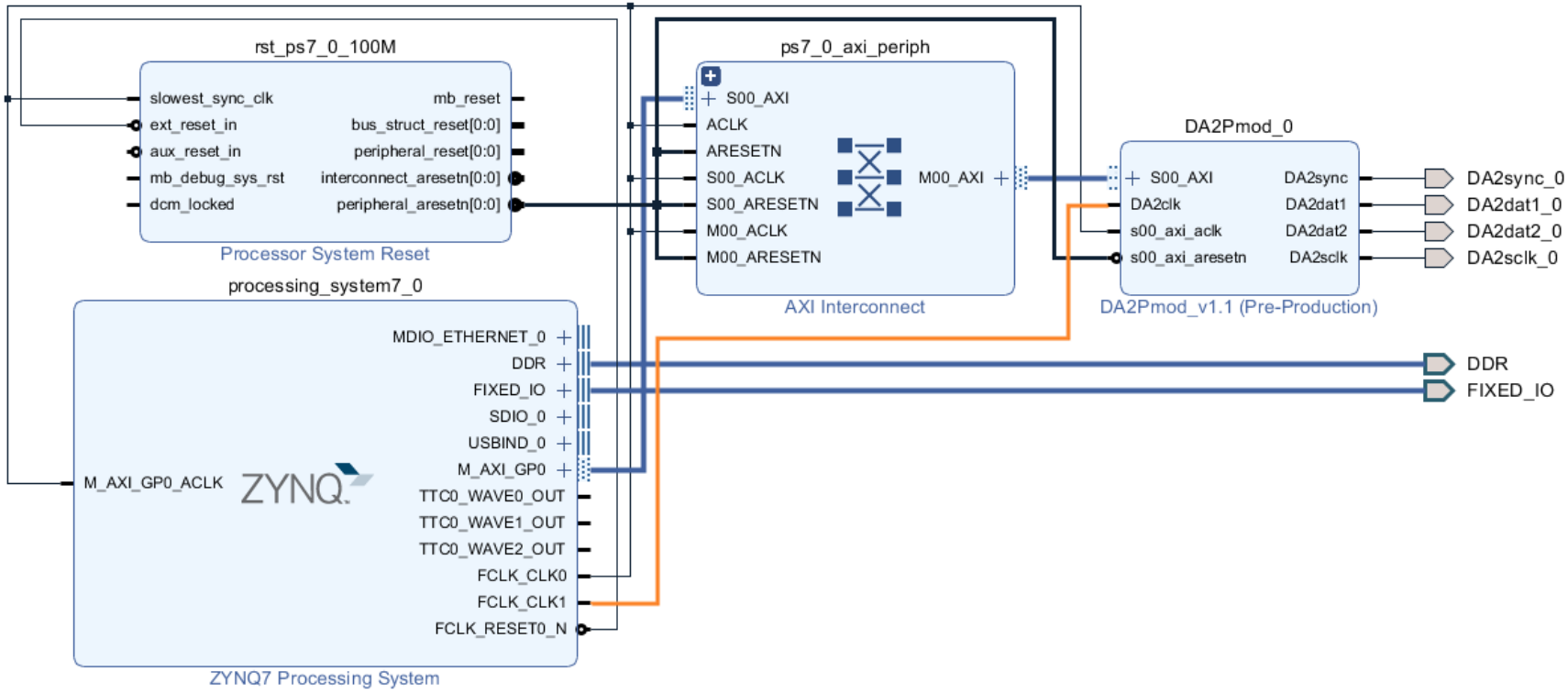
Zynq SPI Peripheral - DAC. The DA2Pmod uses an external 50 MHz clock signal. Right-click on the Zynq PS and select *Customize Block, then* select *Clock Configuration.*



Zynq SPI Peripheral - DAC. Expand *PL Fabric Clocks* and select *FCLK_CLK1* and set to 50 MHz. *OK* to continue.

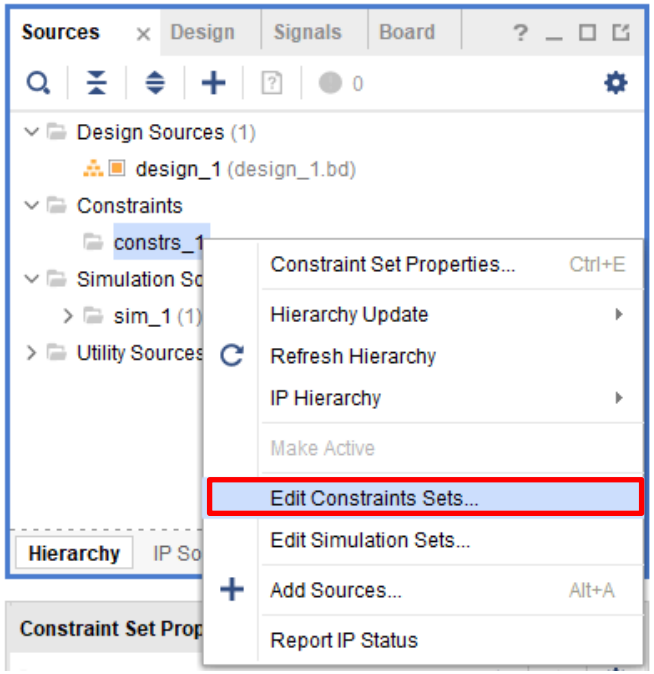


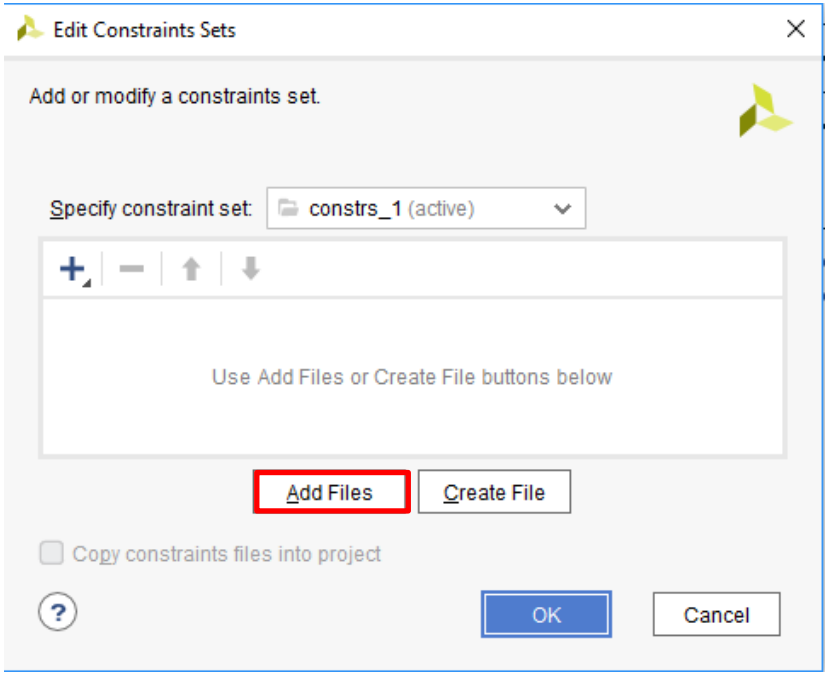
Zynq SPI Peripheral - DAC. The *FCLK_CLK1* port of the Zynq PS is manually connect to the DA2clk port of the DA2Pmod IP.



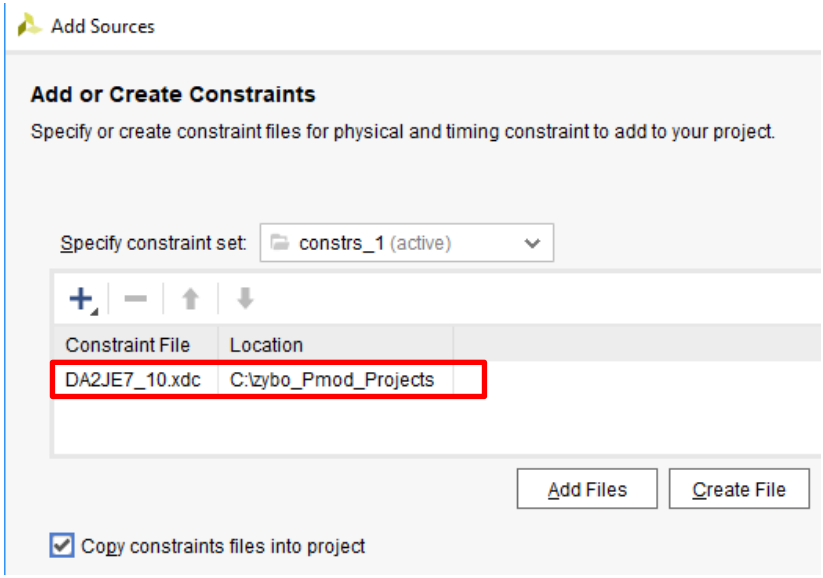
Zynq SPI Peripheral - DAC. An *.xdc* constraint file must be created for this DA2Pmod IP. Copy the *DA2JE7_10.xdc* file to the default directory.

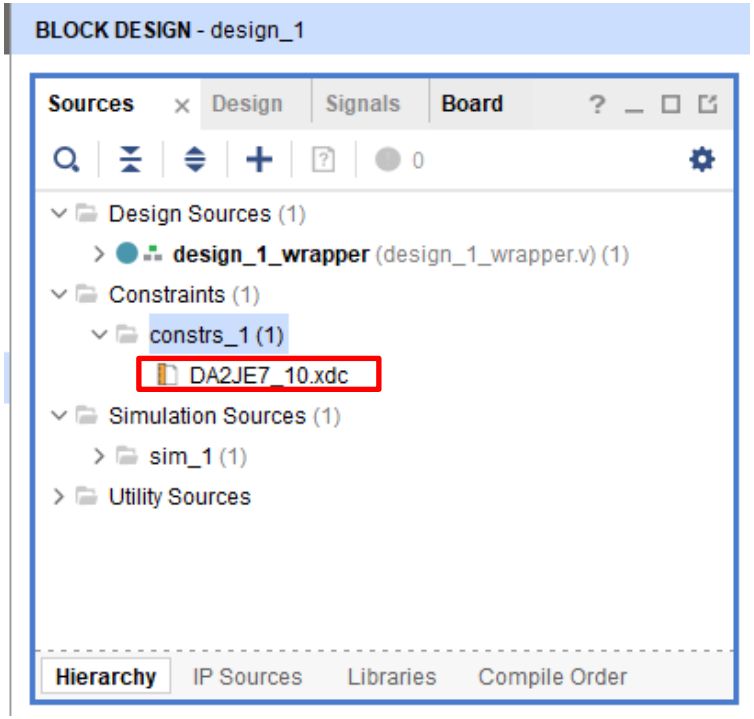
Right-click on *constrs_1* and select *Edit Constraints Sets*. *Select Add Files*.



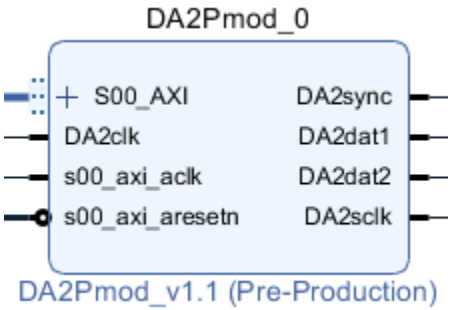


Zynq SPI Peripheral - DAC. The *DA2JE7_10.xdc* constraint file connects the DA2Pmod to Zybo port JE, signal pins 7 to 10. JE pins 11 and 12 are ground and Vcc.





Zynq SPI Peripheral - DAC. Right-click and open the *DA2JE7_10.xdc* constraint file to verify that the JE port is connected to the DA2Pmod signals in use.



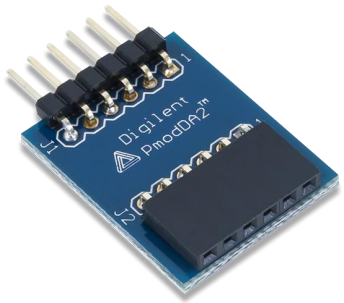
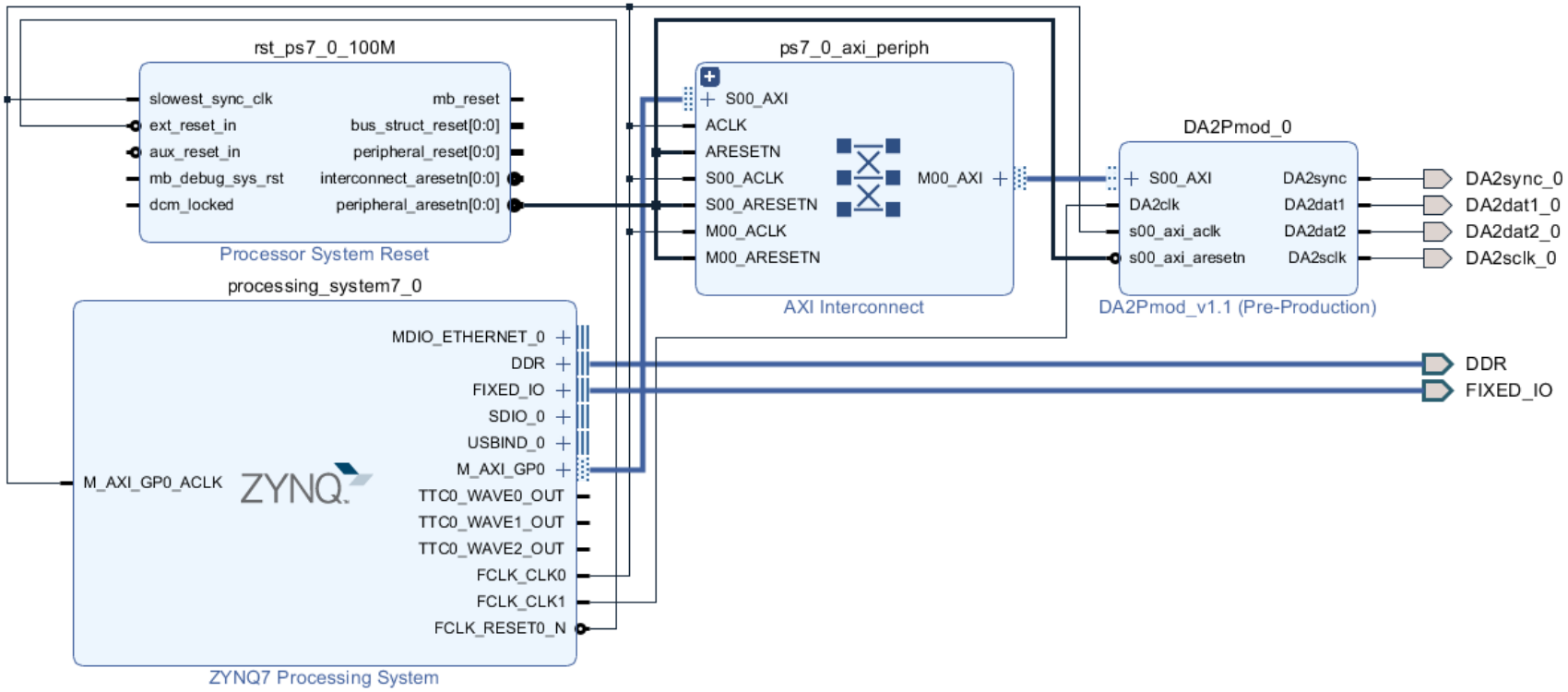


Diagram | Address Editor | DA2JE7_10.xdc

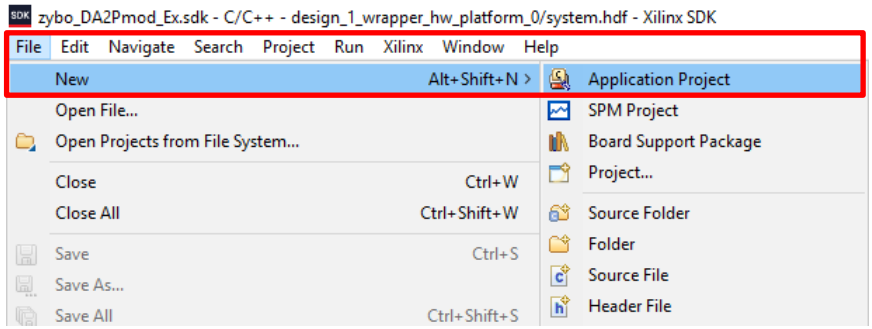
C:/zybo_Pmod_Projects/zybo_DA2Pmod_Ex/zybo_DA2Pmod_Ex.srcs/constrs_1/i

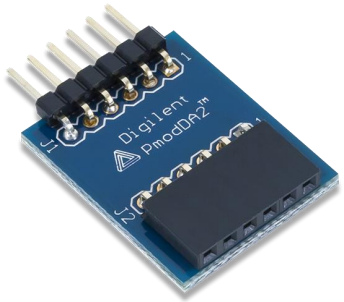
```
166 #set_property PACKAGE_PIN V18 [get_ports {JD10}]
167 #set_property IOSTANDARD LVCMOS33 [get_ports {JD10}]
168
169 #Pmod Header JE
170 #set_property PACKAGE_PIN V12 [get_ports {JE1}]
171 #set_property IOSTANDARD LVCMOS33 [get_ports {JE1}]
172 #set_property PACKAGE_PIN W16 [get_ports {JE2}]
173 #set_property IOSTANDARD LVCMOS33 [get_ports {JE2}]
174 #set_property PACKAGE_PIN J15 [get_ports {JE3}]
175 #set_property IOSTANDARD LVCMOS33 [get_ports {JE3}]
176 #set_property PACKAGE_PIN H15 [get_ports {JE4}]
177 #set_property IOSTANDARD LVCMOS33 [get_ports {JE4}]
178
179 set_property PACKAGE_PIN V13 [get_ports {DA2sync_0}]
180 set_property IOSTANDARD LVCMOS33 [get_ports {DA2sync_0}]
181
182 set_property PACKAGE_PIN U17 [get_ports {DA2dat1_0}]
183 set_property IOSTANDARD LVCMOS33 [get_ports {DA2dat1_0}]
184
185 set_property PACKAGE_PIN T17 [get_ports {DA2dat2_0}]
186 set_property IOSTANDARD LVCMOS33 [get_ports {DA2dat2_0}]
187
188 set_property PACKAGE_PIN Y17 [get_ports {DA2sclk_0}]
189 set_property IOSTANDARD LVCMOS33 [get_ports {DA2sclk_0}]
190
191 #USB-OTG overcurrent detect pin
192 #set_property PACKAGE_PIN U13 [get_ports otg_oc]
193 #set_property IOSTANDARD LVCMOS33 [get_ports otg_oc]
```

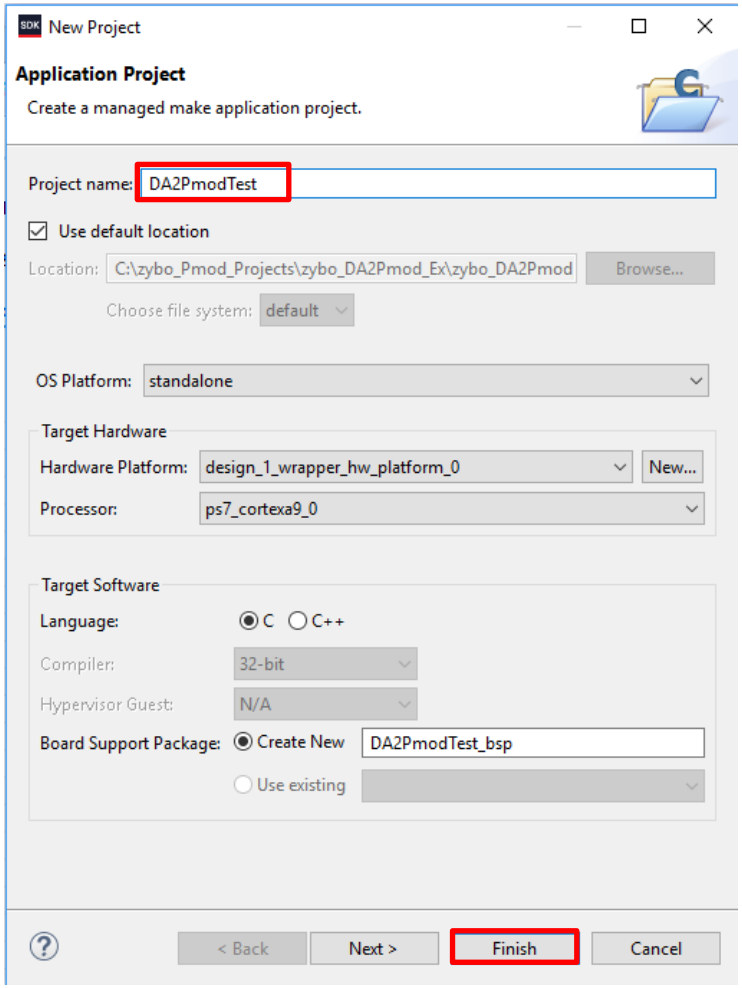
Zynq SPI Peripheral - DAC. Finish the Vivado hardware design by creating the HDL wrapper, generate the bitstream, export hardware (include bitstream) and lauch SDK. SDK imports the hardware specification.



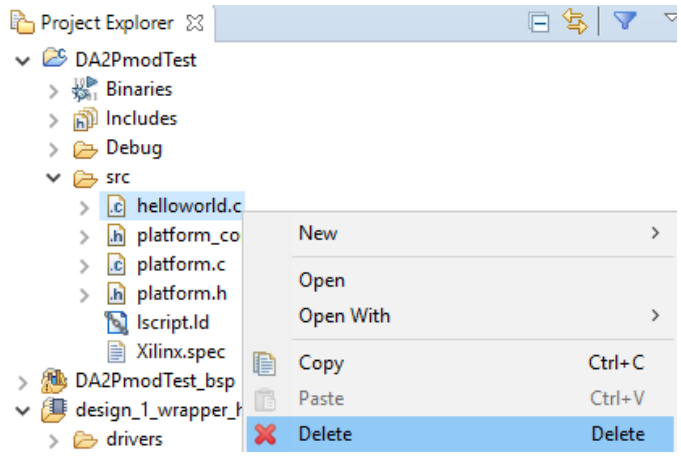
Zynq SPI Peripheral - DAC. In SDK, select *File…New…Application Project* as *DA2PmodTest*. *Finish* to continue.

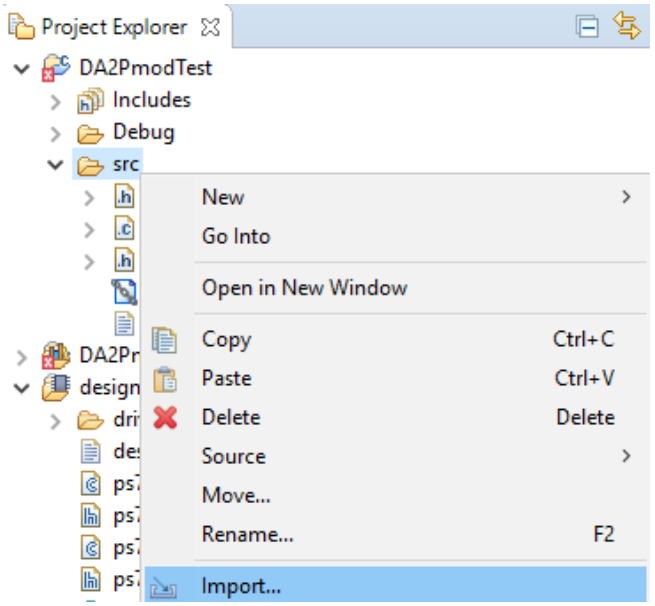


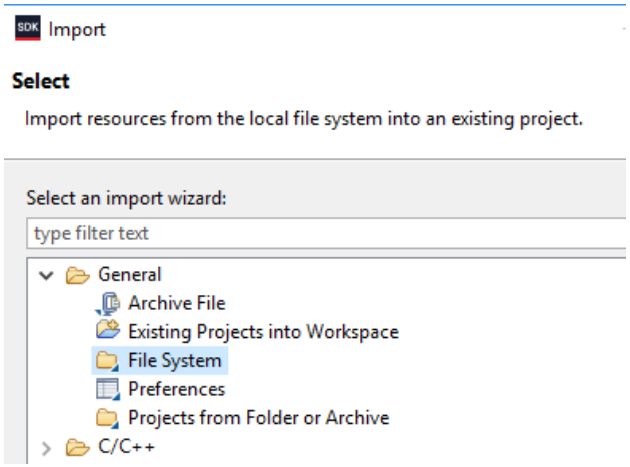


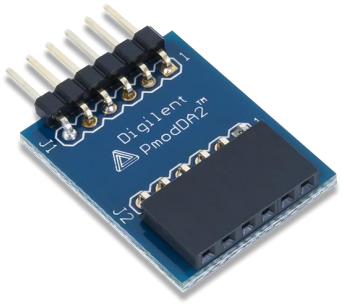


Zynq SPI Peripheral - DAC. Right-click and delete *helloworld.c*. Right-click on *src*…*Import*, select *File System* and browse to the location of *main.c* and select.

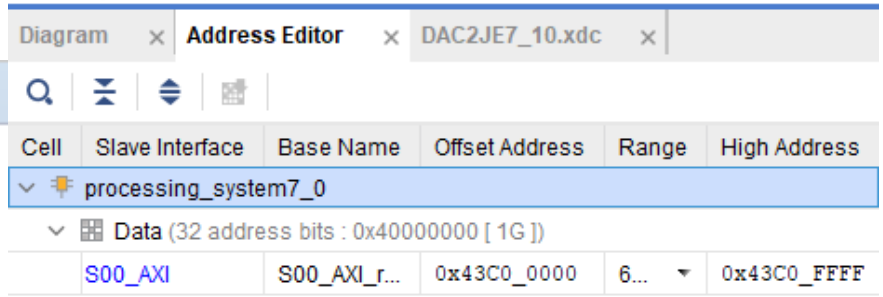








Zynq SPI Peripheral - DAC. Double-click on *main.c*. The DA2Pmod base addresses are in the Address Editor in Vivado and are 32-bit registers (4 bytes).



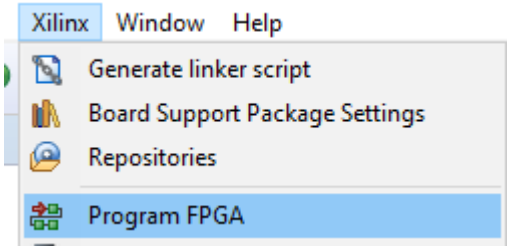
system.hdf system.mss main.c

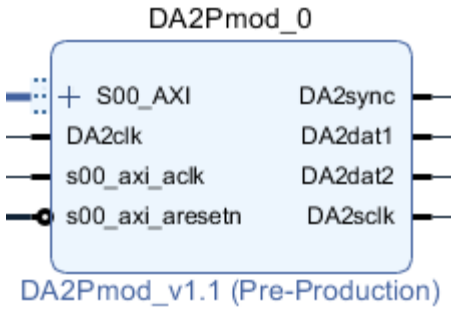
```
// main.c DA2PmodTest   ECE3622
// Dennis Silage   c2019

#include "xparameters.h"
#include "xil_io.h"
#include "xil_printf.h"

//DAC2Pmod from Address Editor in Vivado
#define DA2acq  0x43C00000   //DA2 acquisition    - output
#define DA2dav  0x43C00004   //DA2 data available - input
#define DA2dat1 0x43C00008   //DA2 channel 1 data - output
#define DA2dat2 0x43C0000C   //DA2 channel 2 data - output

int main(void)
{
    int dacdata=0;  //DAC ramp data
    int dacdav;     //DAC data available
    int dacacq=0;   //DAC acquire

    xil_printf("\n\rStarting DA2 Pmod demo test...\n\r");
    Xil_Out32(DA2acq,0);
```

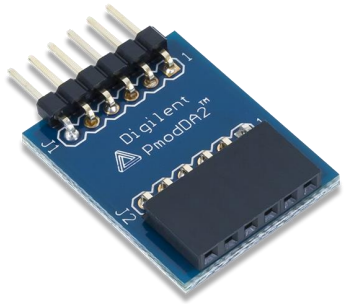
Zynq SPI Peripheral - DAC. The local int *dacacq* is the control signal and the local int *dacdav* is the status return signal. *main.c* generates a linear ramp of data outputted by the PmodDA2.

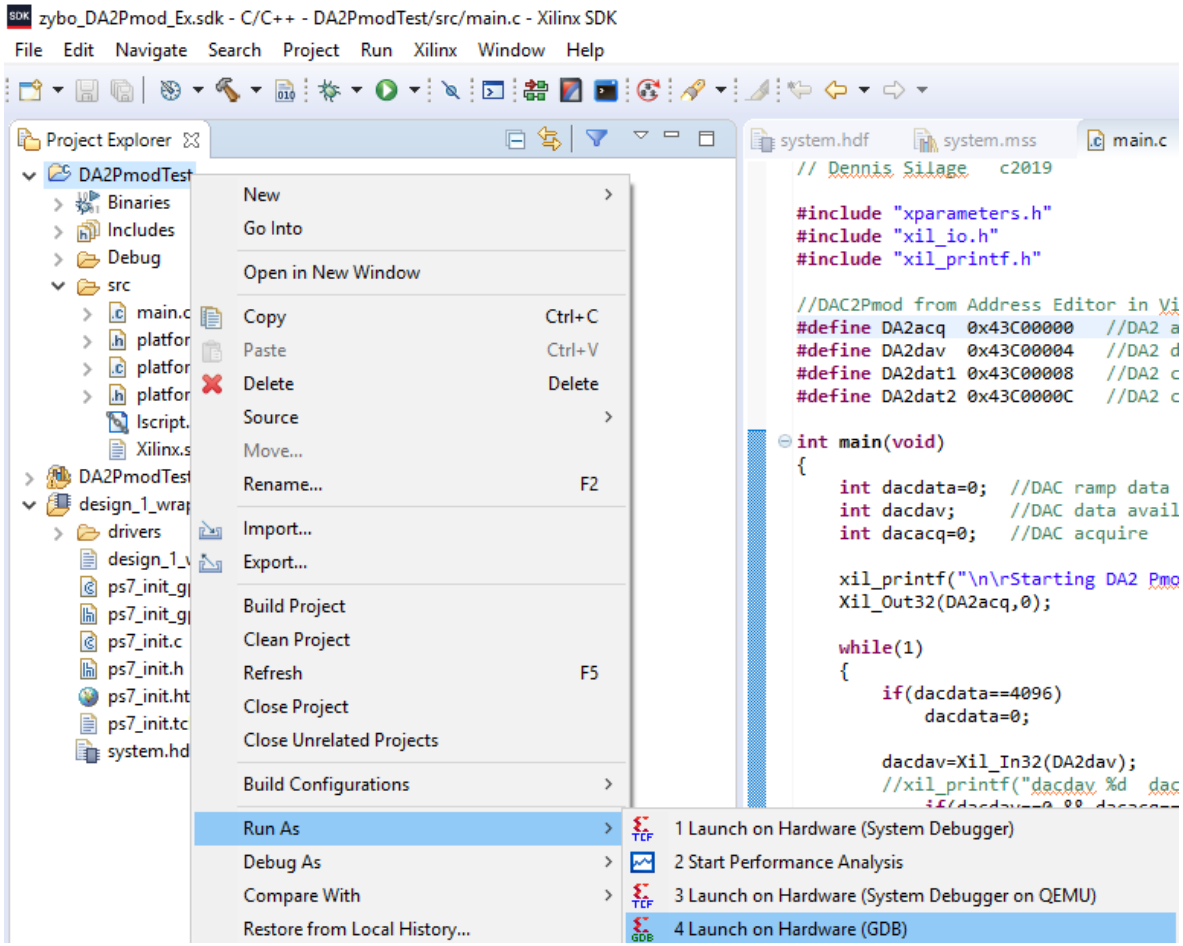
```
    while(1)
    {
        if(dacdata==4096)
            dacdata=0;

        dacdav=Xil_In32(DA2dav);
        //xil_printf("dacdav %d  dacacq %d dacdata %d\n\r",dacdav, dacacq, dacdata);
            if(dacdav==0 && dacacq==0)          //DAC not in use?
                {
                Xil_Out32(DA2dat1, dacdata);    //output DAC data
                Xil_Out32(DA2dat2, dacdata);    //output DAC data
                Xil_Out32(DA2acq,1);            //DAC acquire
                dacacq=1;
                }
            if(dacdav==1 && dacacq==1)          //DAC data available
                {
                Xil_Out32(DA2acq,0);            //stop DAC acquire
                dacacq=0;
                dacdata++;                      //increment DAC data
                }
    }
}
```

Zynq SPI Peripheral - DAC. Program the FPGA and *Run As…4 Launch on Hardware (GDB)*.









Zynq SPI Peripheral - DAC. Connect the Waveforms oscilloscope channel 1 input to the PmodDA2 channel 1 output (pin 1) and the Waveforms oscilloscope ground to PmodDA2 pin 5.

Zynq SPI Peripheral - DAC. The DA2PmodTest project outputs a linear ramp at full resolution of 4096 points in 7.662 msec or ≈1870 nsec/sample. The controller-datapath Verilog module requires 680 nsec.

End of Zynq
SPI Peripherals