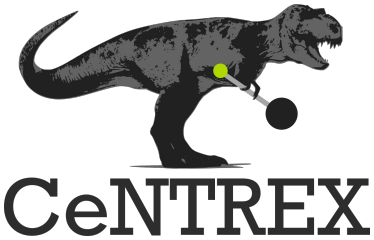
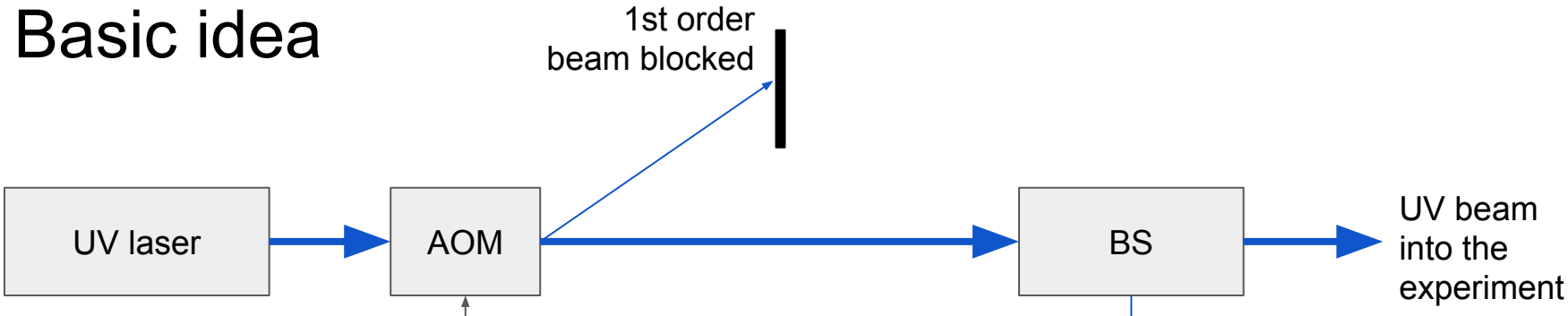
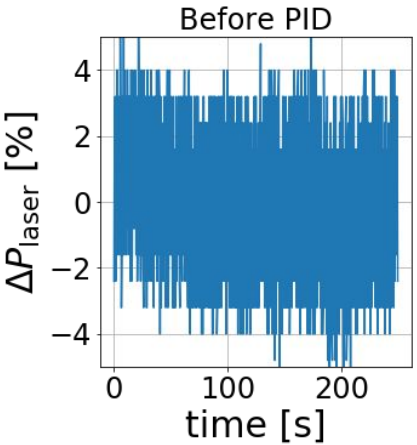


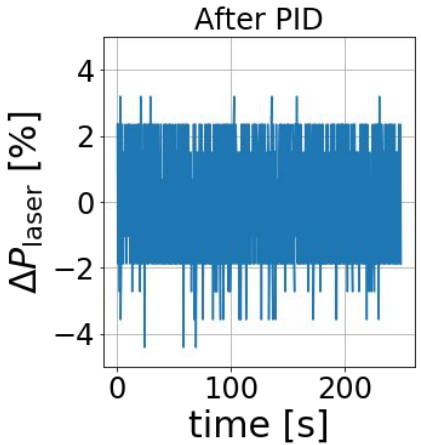

# Laser power PID control

JK

# Basic idea

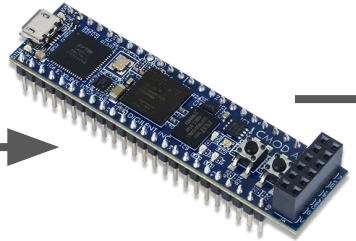

1st order
beam blocked
UV laser
AOM
BS
UV beam
into the
experiment
amp
PID
photodiode
setpoint,
$K_P$, $K_I$, ...
computer


Before PID
$\Delta P_{laser}$ [%]
4
2
0
−2
−4
0
100
200
time [s]

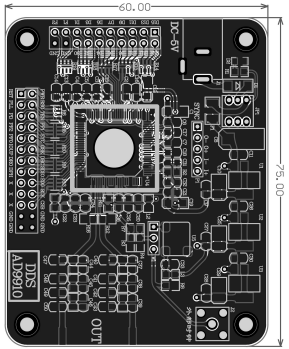

After PID
$\Delta P_{laser}$ [%]
4
2
0
−2
−4
0
100
200
time [s]

# Hardware overview


**amplified detector**
Thorlabs PDA25K2



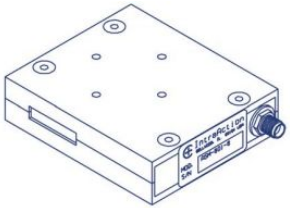
**FPGA board**
Digilent Cmod A7




**DDS board**
with AD9910

**power amp**
Mini-Circuits ZHL-5W-1+

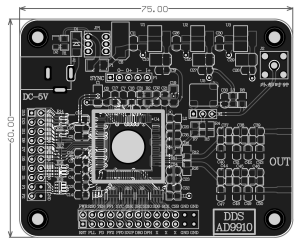

**µC board**
PJRC Teensy 3.6



**AOM**
IntraAction ASM-583B45

# Presentation outline

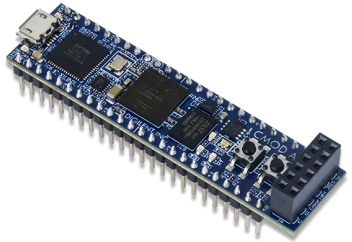
## (1)




**Direct Digital Synthesis**

- intro to DDS
- intro to AD9910
- serial programming
- parallel interface

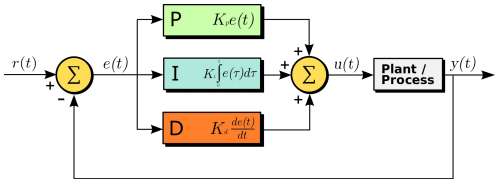
## (2)



**FPGA programming**

- Artix 7, Cmod A7
- Vivado & Basic IO
- Clocking, ADC
- 2s complement
- RS232

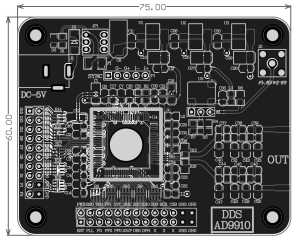
## (3)



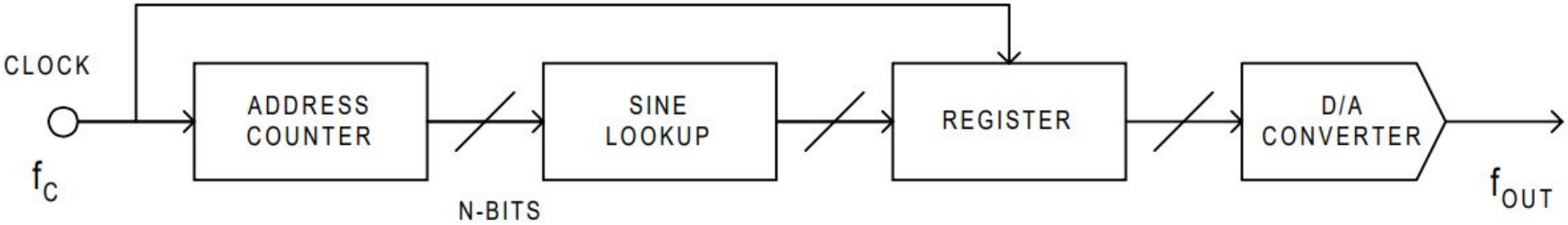

**PID control**

- PID control
- *R*–2*R* network
- DDS output power
- Laser power

# (1) Direct Digital Synthesis




Fixed-frequency DDS:

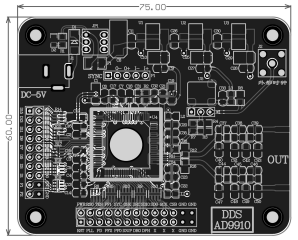


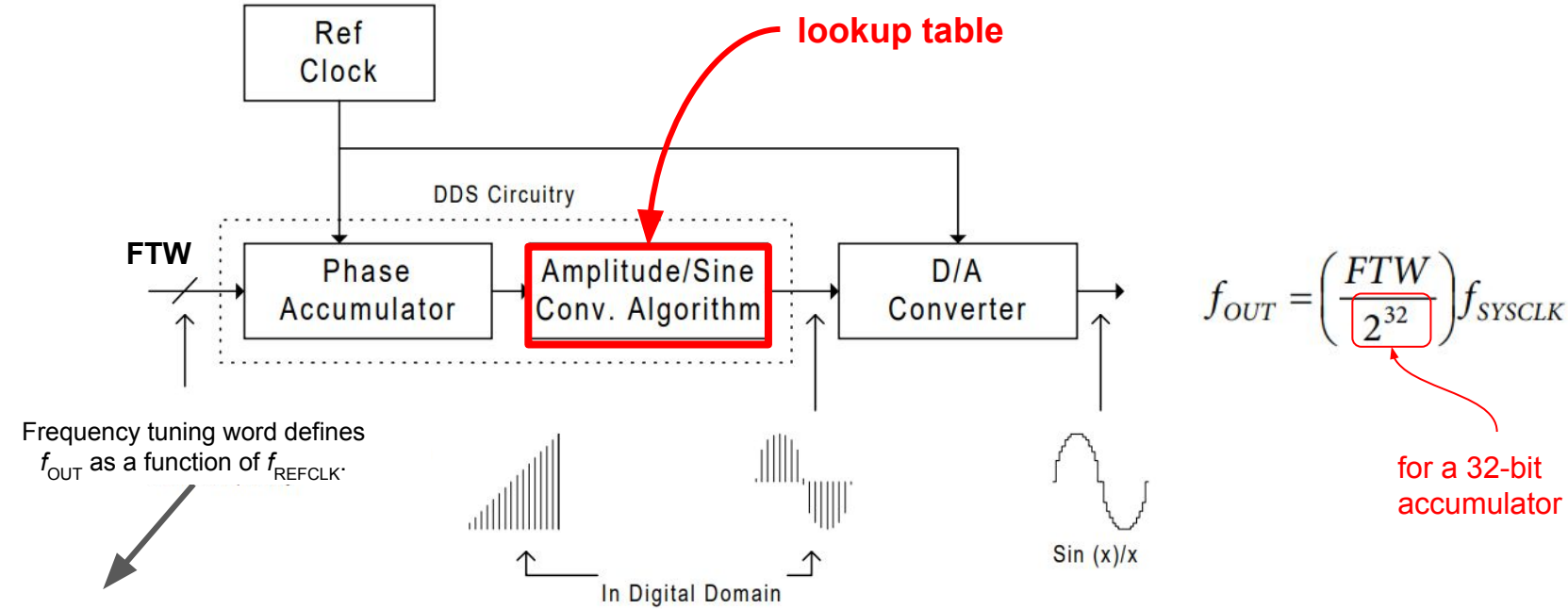

[Fig. 1-2 from Analog Devices: *A Technical Tutorial on Digital Signal Synthesis*, 1999 from https://www.ieee.li/pdf/essay/dds.pdf on 8/26/2019]

**Nyquist theorem:**
Analog signal @ $f_{analog}$ can be reconstructed from a digital signal sampled at $2f_{analog}$.

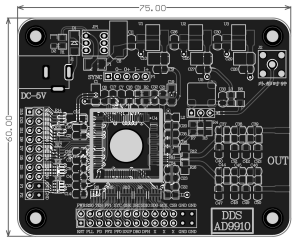
$\rightarrow$ $f_{OUT} < f_C / 2$ $\rightarrow$ jitter($f_{out}$) < jitter($f_C$)

# (1.1) Freq.-agile DDS

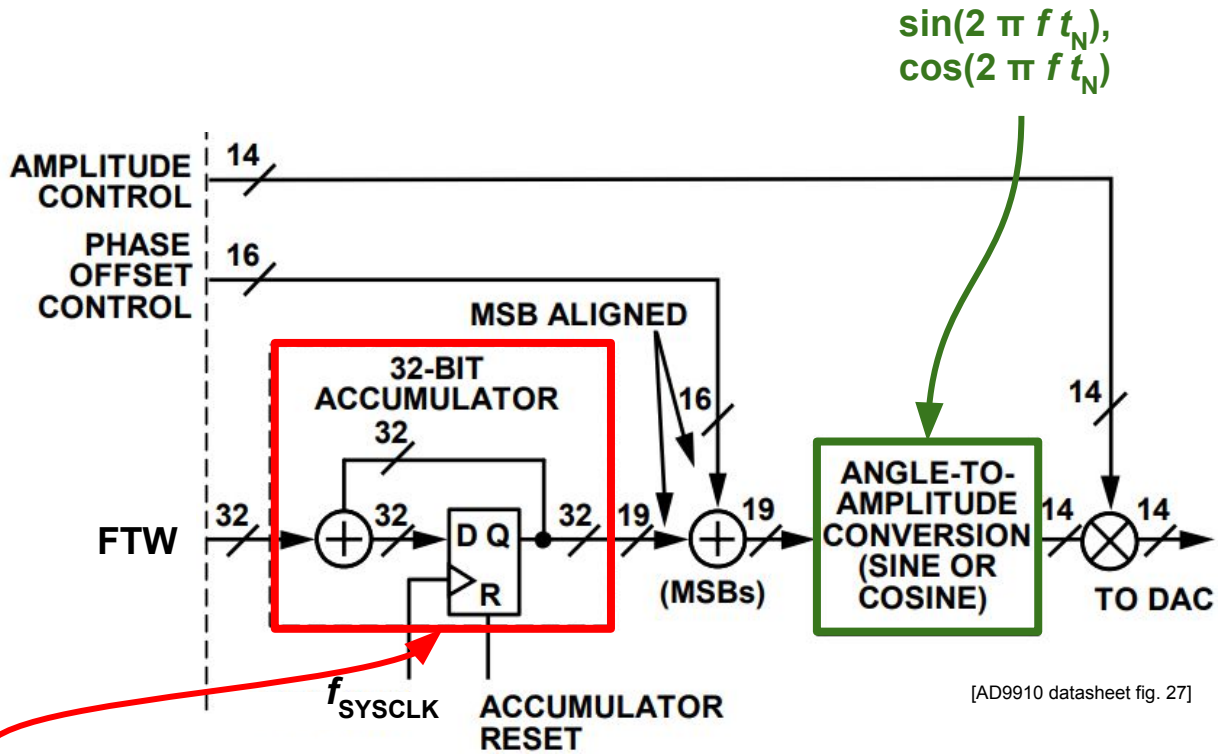





[Fig. 1-4 from Analog Devices: *A Technical Tutorial on Digital Signal Synthesis*, 1999 from https://www.ieee.li/pdf/essay/dds.pdf on 8/26/2019]



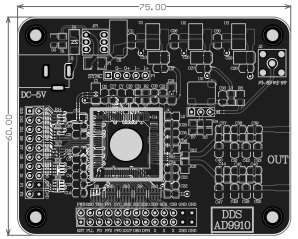
# (1.2) AD9910 DDS core




[AD9910 datasheet fig. 27]

$t_N = (\text{FTW} \cdot N) \bmod 32$

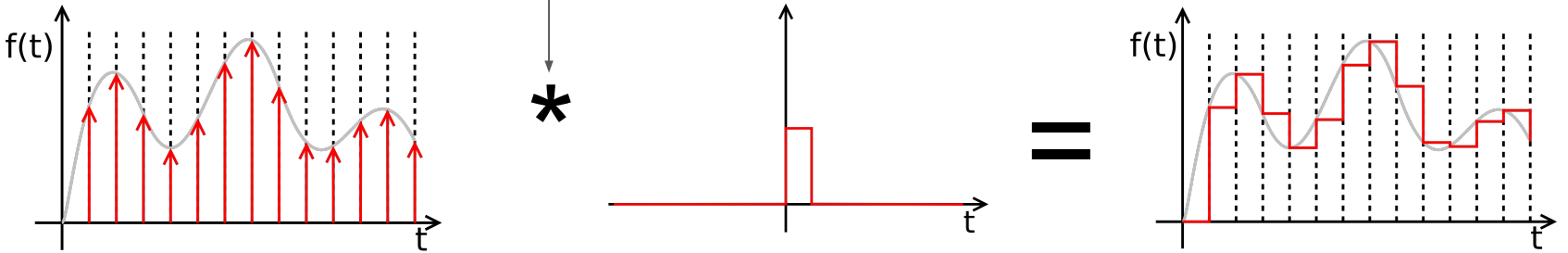
# (1.3) DDS signal reconstitution




digital samples =
modulated Dirac comb

convolution

“zero-order hold” =
piecewise-linear output




**Convolution theorem**

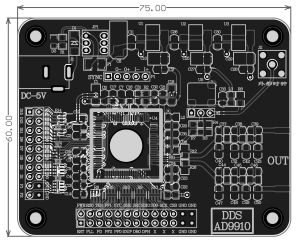
$$\mathcal{F}\{ f(t) \} = \mathcal{F}\{ f(t) \} \cdot \mathcal{F}\{ \text{rect} \}$$

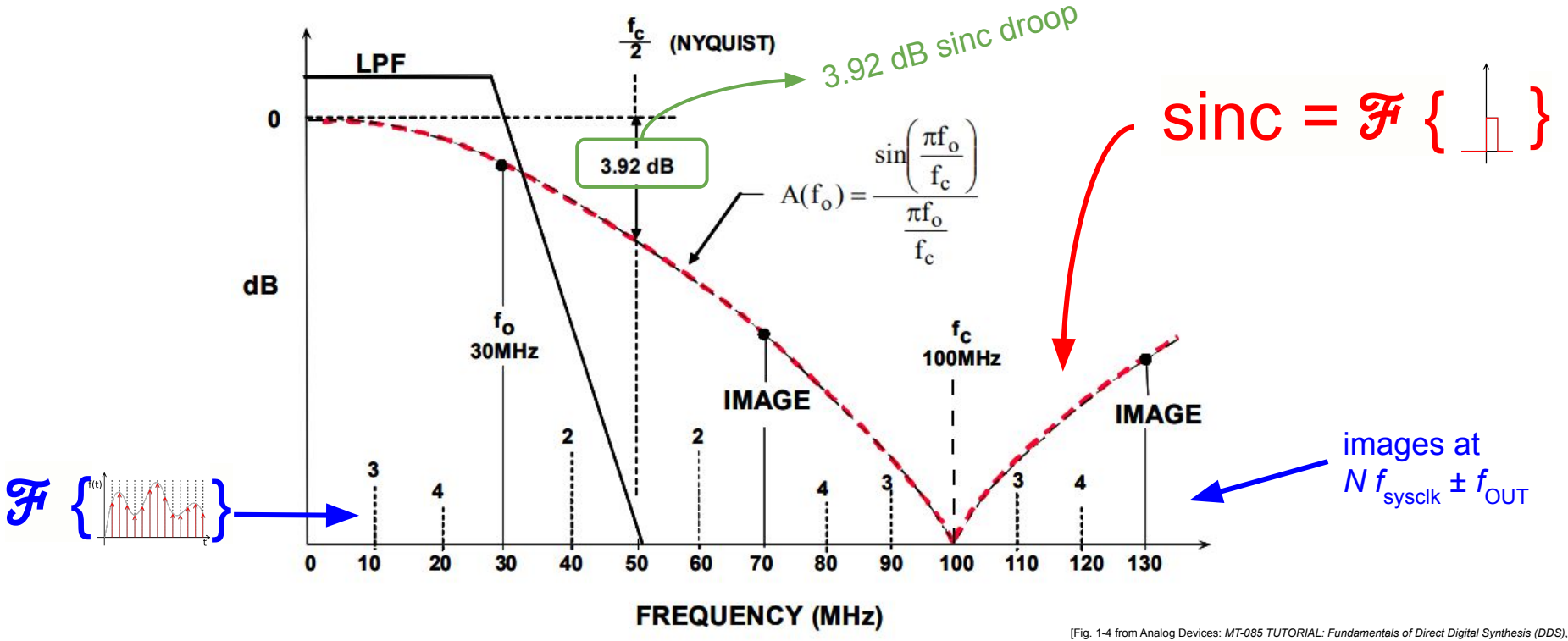
desired output + aliased images

sin(x) / x envelope

[Figures from Wikipedia: *Zero-order hold* from https://en.wikipedia.org/wiki/Zero-order_hold on 8/27/2019]

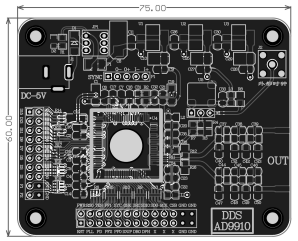
# (1.4) DDS output spectrum

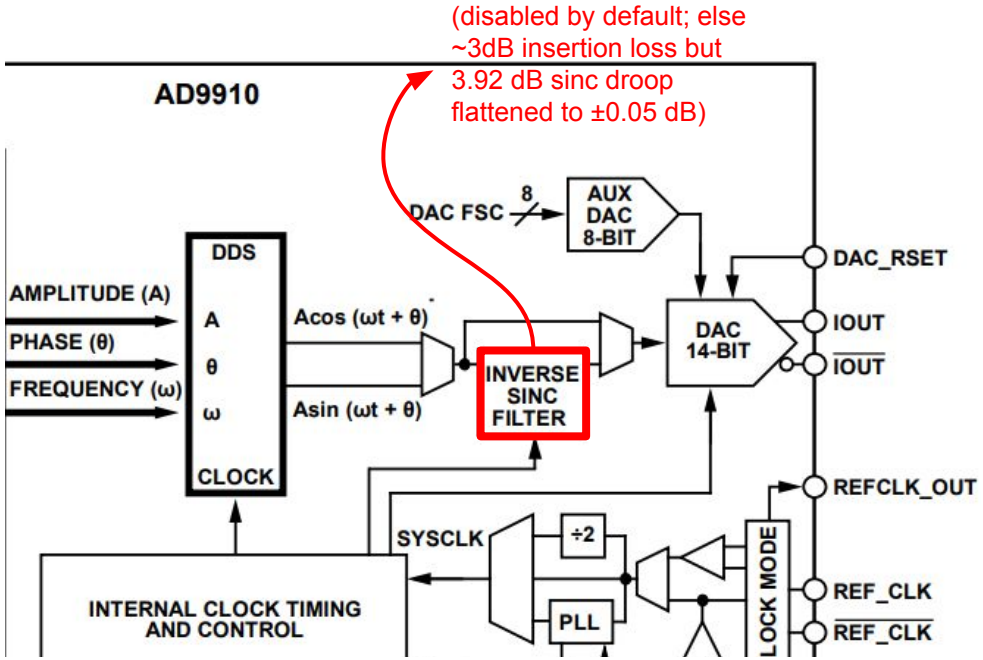





[Fig. 1-4 from Analog Devices: *MT-085 TUTORIAL: Fundamentals of Direct Digital Synthesis (DDS)*, 2008 from https://www.analog.com/media/en/training-seminars/tutorials/MT-085.pdf on 8/27/2019]

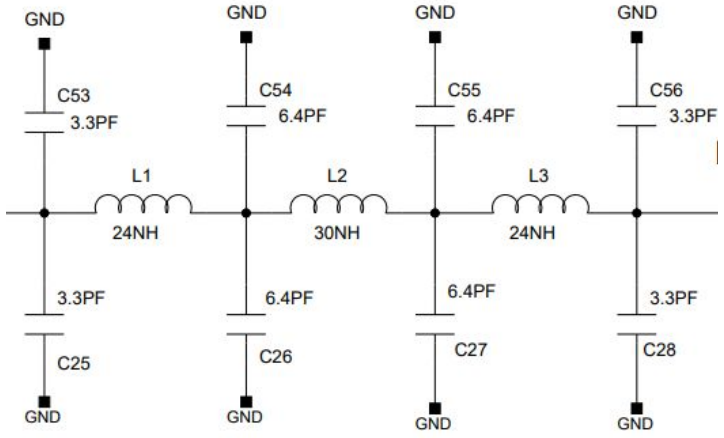
# (1.5) AD9910 sinc & aliasing filter






$$< \frac{1\ \text{GHz}}{2} = \frac{\text{max. SYSCLK}}{2}$$

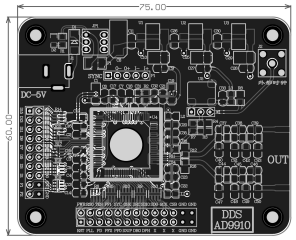
400 MHz Low Pass Filter




[AD9910 datasheet fig. 2]

[from official AD9910 eval board schematic]

# (1.6) Serial programming AD9910



All configuration registers:

Inverse sinc filter enable

Parallel data port enable

REFCLK input divider bypass

0: 00000000,0**0**000000,00000000,00000000,
1: 00000000,01000000,00001000,001**0**0000,
2: 00011111,00111111,**0**1000000,00000000,
3: 00000000,00000000,01111111,01111111,
4: 11111111,11111111,11111111,11111111,
5: 11101001,00100000,01100100,10001001,11000101,11100000,
6: 10000111,00101000,00000101,10101001,10000000,00101000,
7: 00000000,00000000,00000000,00000000,
8: 00000000,00000000,
9: 00000000,00000000,00000000,00000000,
A: 00000000,00000000,00000000,00000000,
B: 10100000,10011001,11100100,00011010,00011001,00101010,00000011,11000100,
C: 11000100,10100000,10110001,10000101,00000011,11000000,10000011,10110000,
D: 01001110,11010001,00000000,00011100,
E: 00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,
F: 00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,
10: 00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,
11: 00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,
12: 00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,
13: 00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,
14: 00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,
15: 00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,
16: 00111000,11010000,01010010,11000111,

**8 profiles**

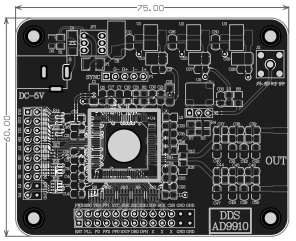
amplitude phase frequency

SPI


Teensy AD9910

Example C and Python code for SPI programming available at:
https://github.com/js216/Arduino-codes/tree/master/Chinese_AD9910

# (1.7) AD9910 parallel interface




| F[1:0] | D[15:0] | Parameter(s) |
|---|---|---|
| 00 | D[15:2] | 14-bit amplitude parameter (unsigned integer) |
| 01 | D[15:0] | 16-bit phase parameter (unsigned integer) |
| 10 | D[15:0] | 32-bit frequency parameter (unsigned integer) |
| 11 | D[15:8] | 8-bit amplitude (unsigned integer) |
| | D[7:0] | 8-bit phase (unsigned integer) |

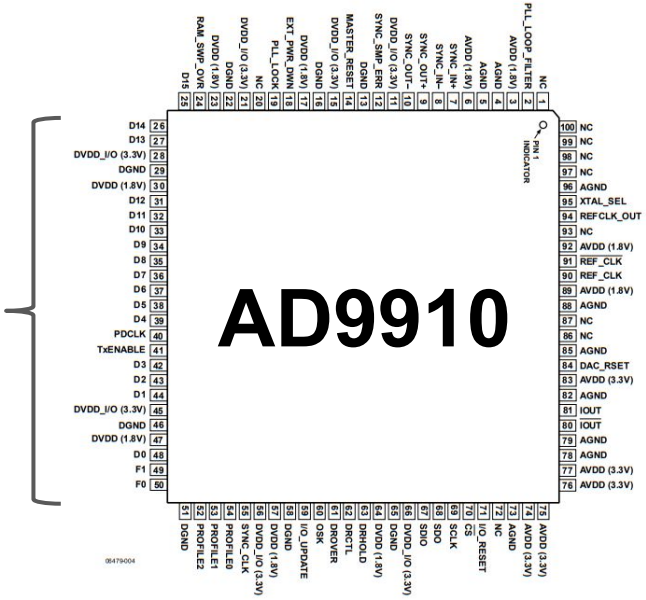
16-bit parallel data input

**D**[15:0]

&

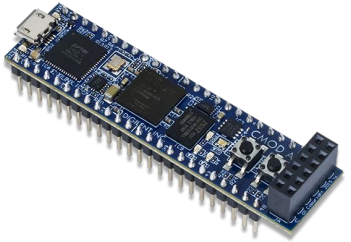
**F**[1:0]

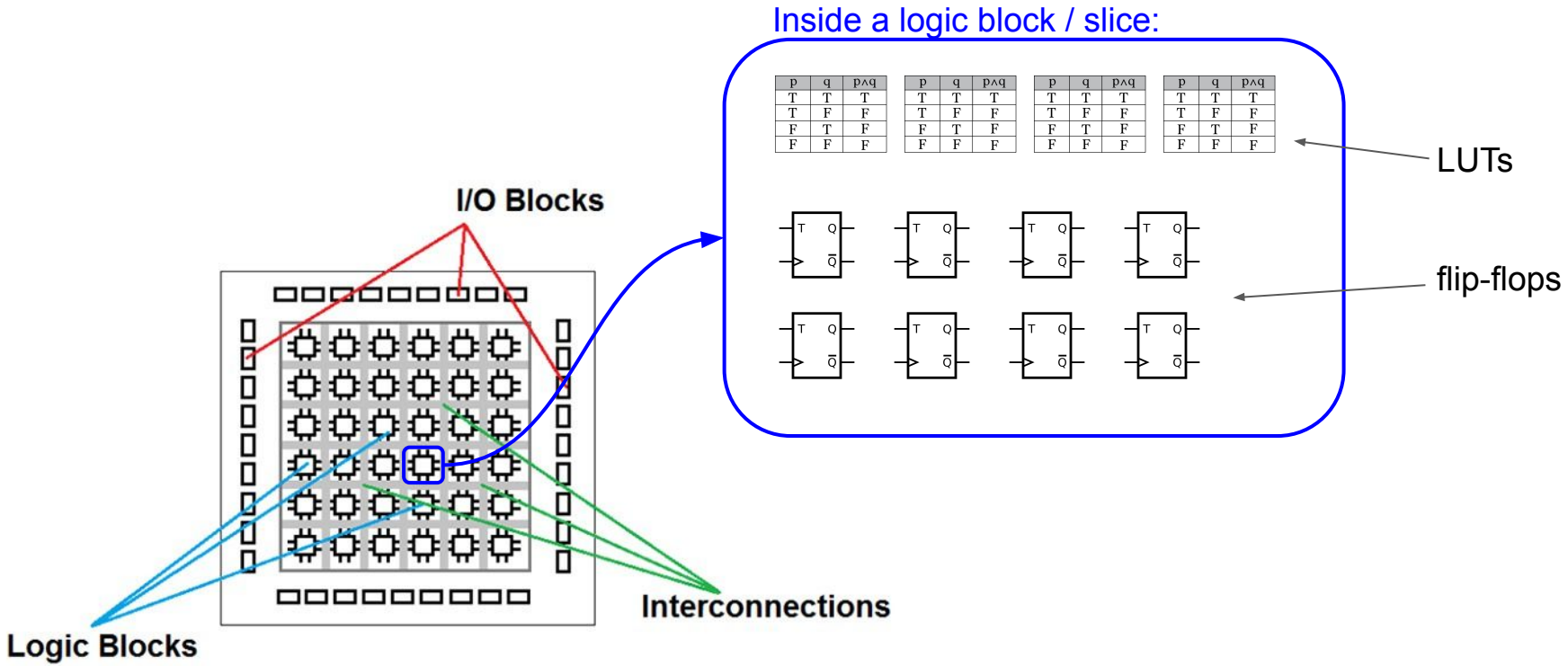
Parallel Port Destination Bits



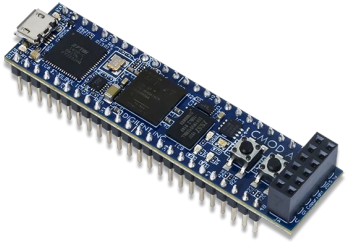

[AD9910 datasheet fig. 5]

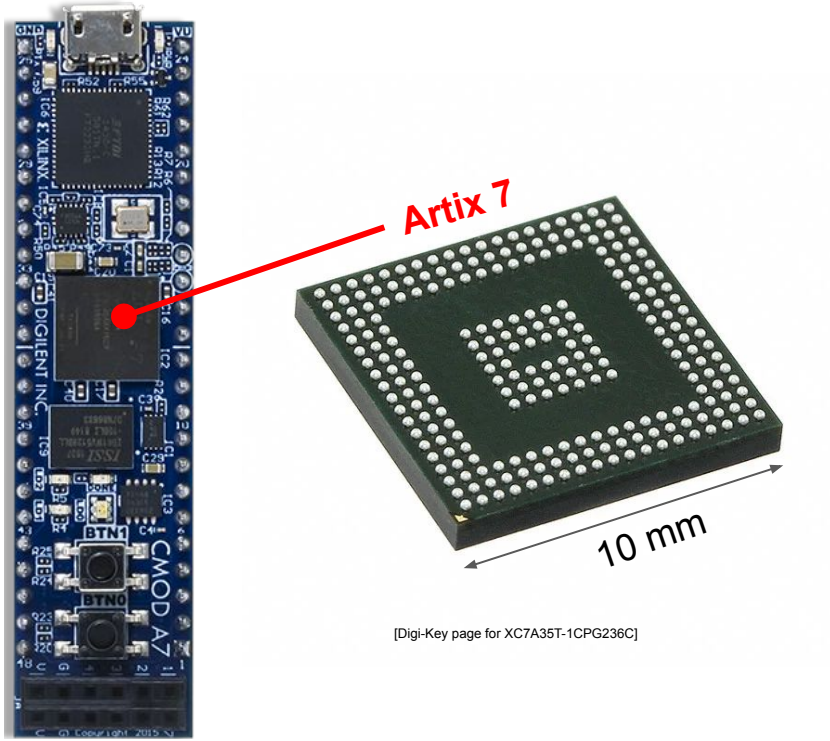
# (2) FPGA basics





# (2.1) Cmod A7 board






[Digi-Key page for XC7A35T-1CPG236C]

## Artix 7 (XC7A35T) features:

- **5200 logic slices** w/ 4 LUTs and 8FFs each
- **90 DSP slices** with pre-adder, a 25 x 18 multiplier, an adder, and an accumulator
- **225 kB of Block RAM**
- **5 clock management tiles** (generates multiples of system clock)
- **dual 12-bit 1 MSPS ADC**

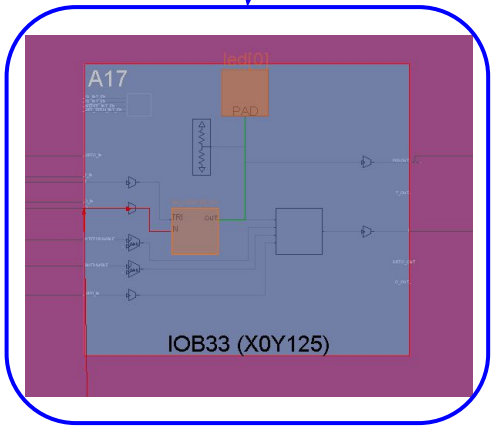
https://store.digilentinc.com/cmod-a7-breadboardable-artix-7-fpga-module/

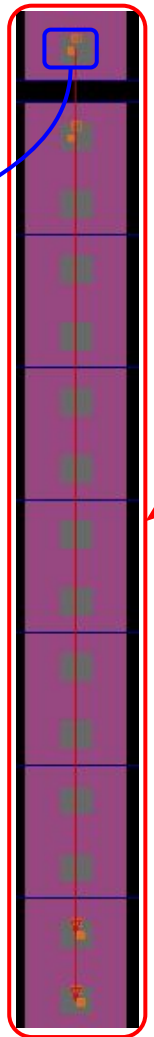
# (2.2) FPGA programming

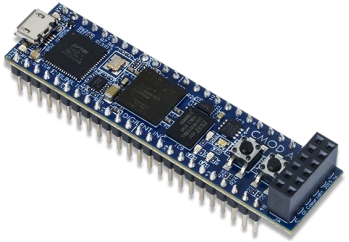
Verilog code:

```
module btn_led(
  input [1:0]btn,  // buttons
  output [1:0]led  // LEDs
);

  assign led[1] = btn[0];
  assign led[0] = btn[1];

endmodule
```

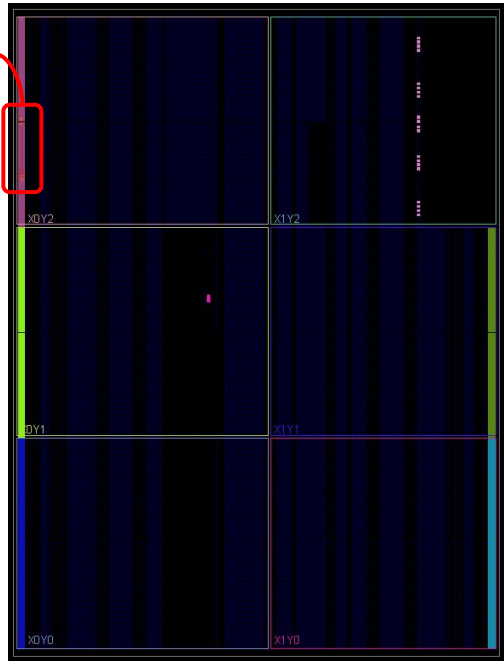
Code available at:
https://github.com/js216/FPGA-codes/blob/master/1%20btn_led/project_1.srcs/sources_1/imports/cmoda7btn_demo/btn_led.v



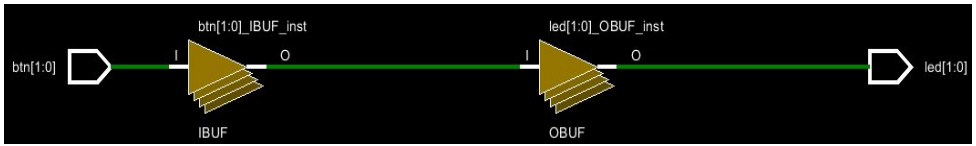





“Synthesized” design:




“Elaborated” design:



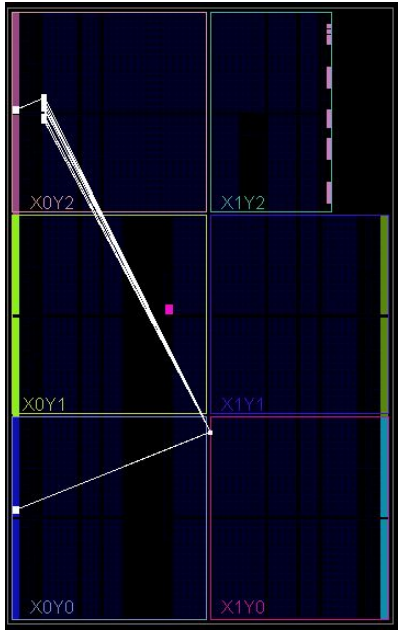
# (2.3) Blinking LED

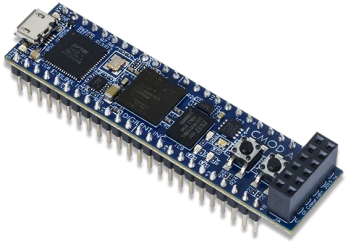
```
module blink(
 input sysclk,
 output led,
);

reg [22:0]counter;
always @(posedge sysclk)
 counter <= counter + 1;

assign led = counter[22];

endmodule
```

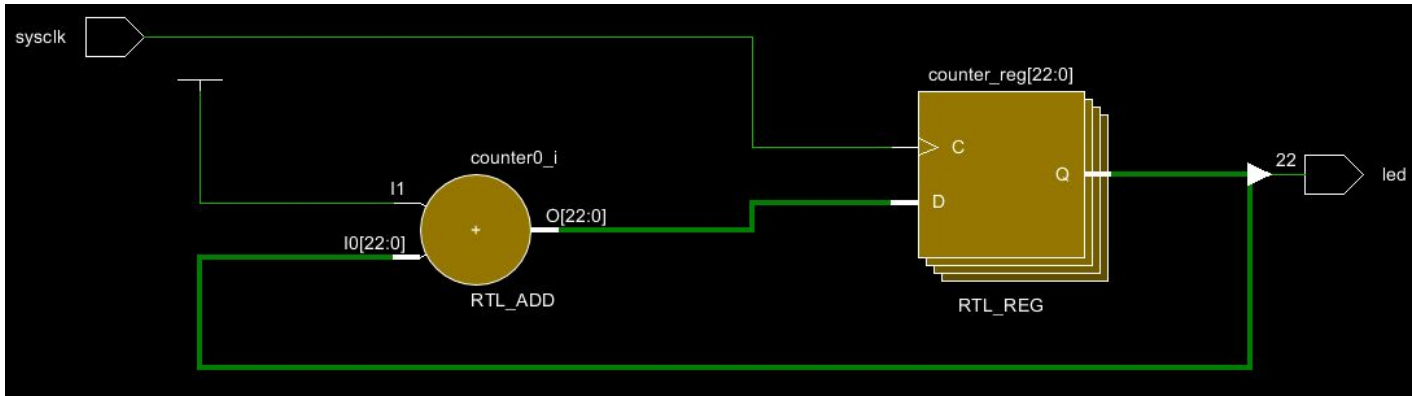
define inputs / outputs

23-bit counter to divide
12 MHz system clock to ~1.4 Hz

counter MSB controls the LED

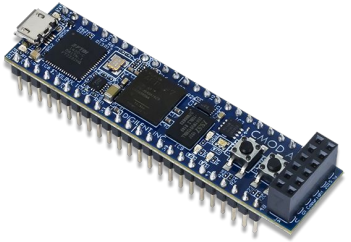





Code available at:
https://github.com/js216/FPGA-codes/blob/master/2%20blink/blink.srcs/sources_1/new/blink.v

# (2.4) Instantiating IP cores



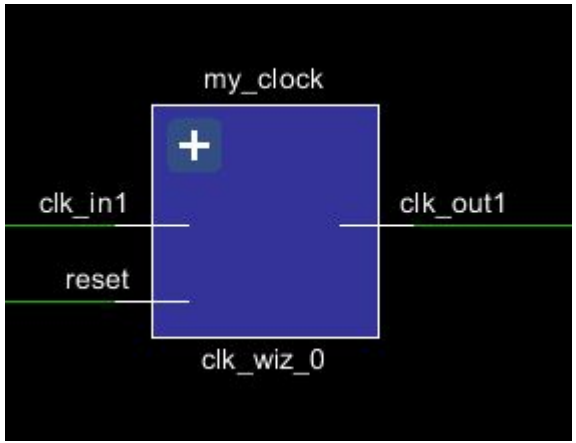
```
// the XADC requires a 100 MHz clock
clk_wiz_0 my_clock (
   .clk_in1(sysclk),
   .clk_out1(clock_100MHz),
   .reset(1'b0)
);
```

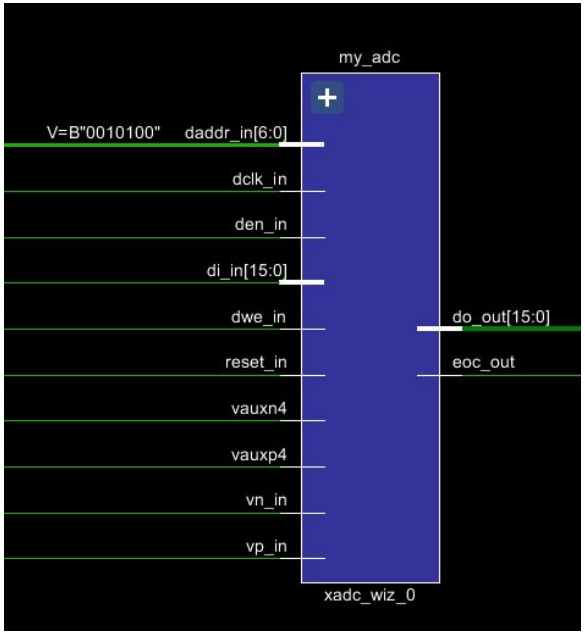
wire clock_100MHz;

```
wire [15:0] data;
wire enable;
xadc_wiz_0 my_adc (
      .daddr_in(8'h14),
      .dclk_in(clock_100MHz),
      .den_in(enable),
      .di_in(0),
      .dwe_in(0),
      .busy_out(),
      .channel_out(),
      .do_out(data),
      .eoc_out(enable),
      .vp_in(vp_in),
      .vn_in(vn_in),
      .reset_in(0),
      .vauxp4(xa_p),
      .vauxn4(xa_n)
);
```

digital data out

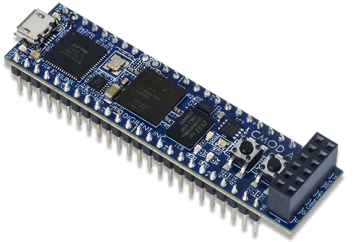
analog input



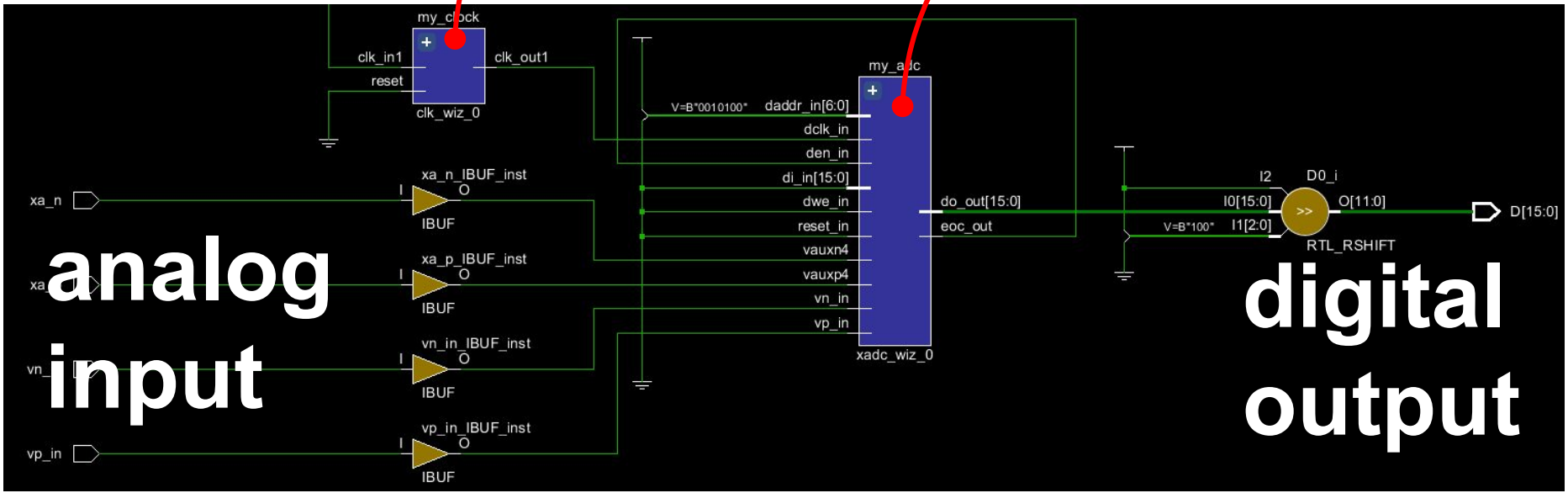

Code available at: https://github.com/js216/FPGA-codes/blob/master/3%20ADC%20test/project_2.srcs/sources_1/imports/project_2/XADCdemo.v

# (2.5) ADC test



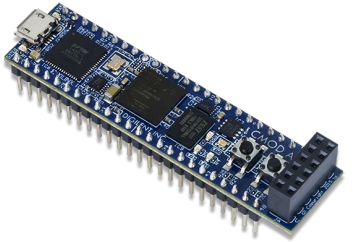
**clock "IP" core:** makes a PLL clock synthesizer

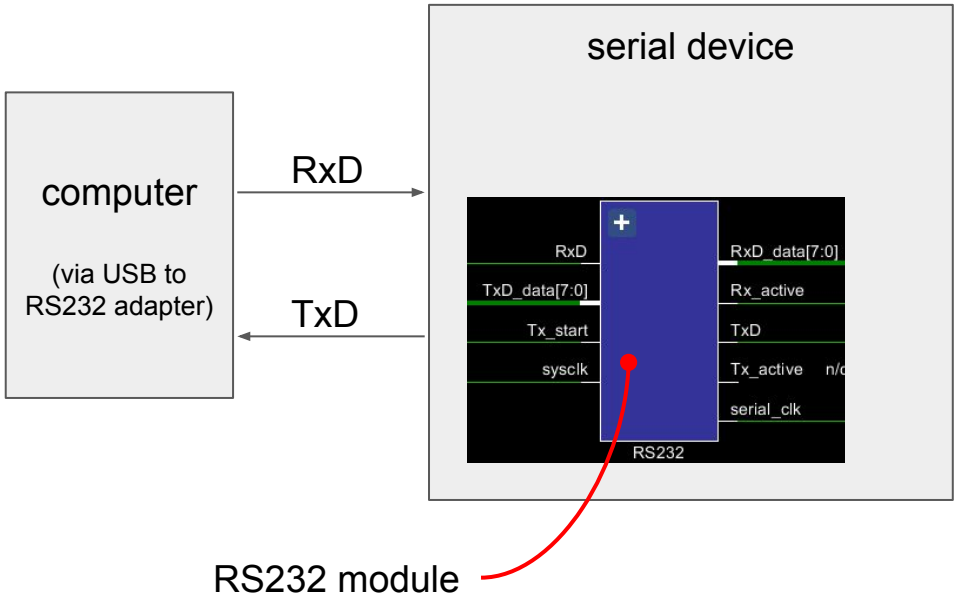
**XADC "IP" core:** makes an ADC block

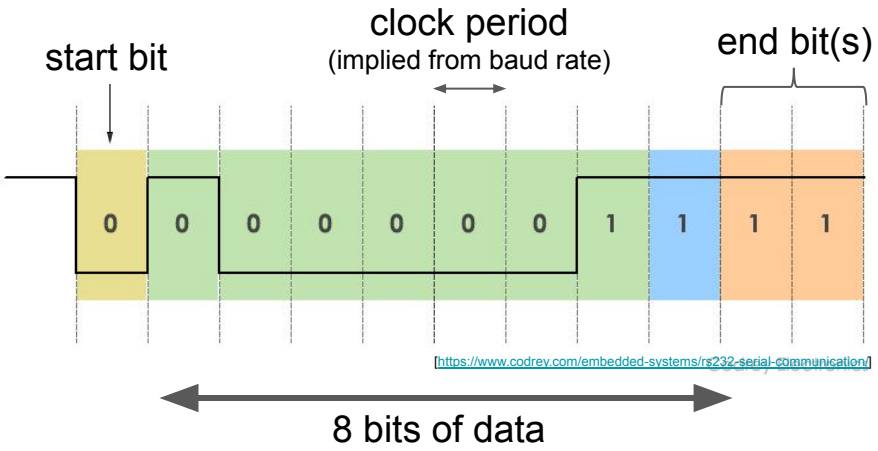



Code available at: https://github.com/js216/FPGA-codes/blob/master/3%20ADC%20test/project_2.srcs/sources_1/imports/project_2/XADCdemo.v

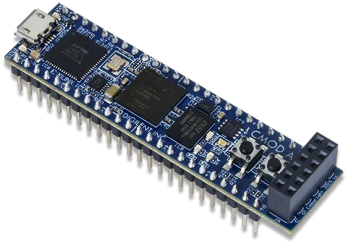
# (2.6) RS232 protocol

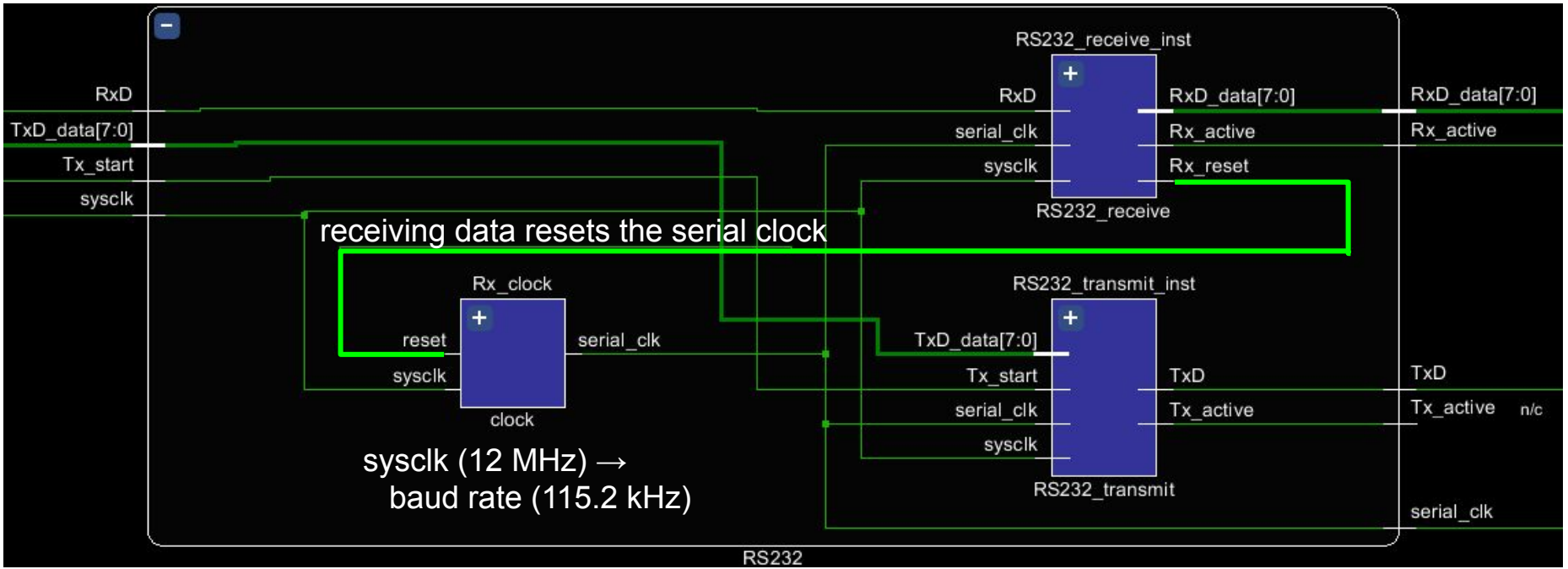




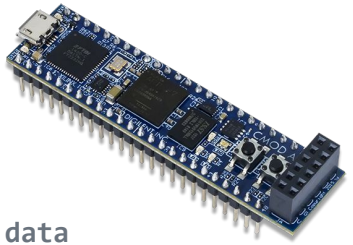



# (2.7) Inside the RS232 module



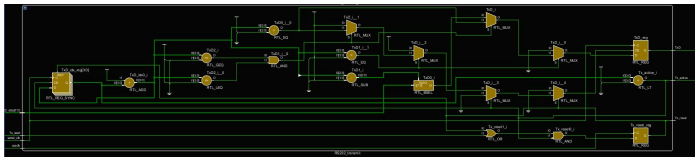


# (2.8) RS232 data transmit

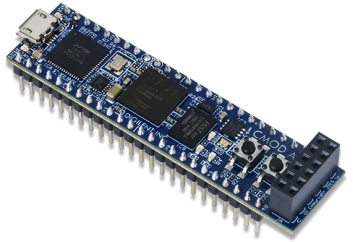


```
module RS232_transmit(
 input sysclk,
 input serial_clk,

 output reg TxD,
 input [7:0]TxD_data,

 input Tx_start,
 output Tx_active,
 output reg Tx_reset
);

// transmission data index
reg [3:0]TxD_idx = 10;
assign Tx_active = TxD_idx < 11;

// detect start of transmission
always @(posedge sysclk)
 Tx_reset = (Tx_reset || Tx_start) &&
~Tx_active;
```

```
always @(posedge serial_clk) begin
 // if beginning transmission, reset data
index
 if (Tx_reset)
   TxD_idx = 0;

 else if (Tx_active) begin
   // increment data index
   TxD_idx = TxD_idx + 1;

   // transmit the data
   case (TxD_idx) inside
       1: TxD = 0; // start bit = 0
   [2:9]: TxD = TxD_data[TxD_idx-2];
      10: TxD = 1; // stop bit(s) = 1
   endcase
 end
end

endmodule
```

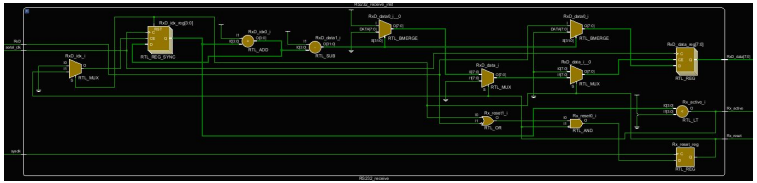


# (2.9) Receive RS232 data

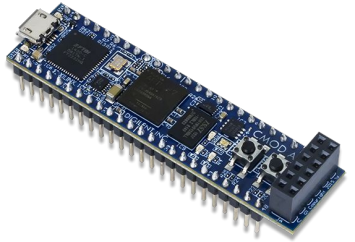


```
module RS232_receive(
 input sysclk,
 input serial_clk,

 input RxD,
 output reg [7:0]RxD_data,

 output Rx_active,
 output reg Rx_reset
);

// data index
reg [3:0] RxD_idx = 10;
assign Rx_active = RxD_idx < 10;

// detect start of data reception
always @(posedge sysclk) begin
 Rx_reset = (Rx_reset || ~RxD) && ~Rx_active;
end
```

```
// receive data
always @(posedge serial_clk) begin
 if (Rx_reset)
   RxD_idx <= 0;
 else if (Rx_active) begin
   RxD_idx = RxD_idx + 1;
   RxD_data[RxD_idx-1] = RxD;
 end
end


endmodule
```



# (2.10) Negative numbers in binary



**2s complement representation**

25 as 8-bit binary number:

```
0001 1001
```

**negation:**
invert and add 1

−25 as 8-bit binary number:

```
1111 0111
```

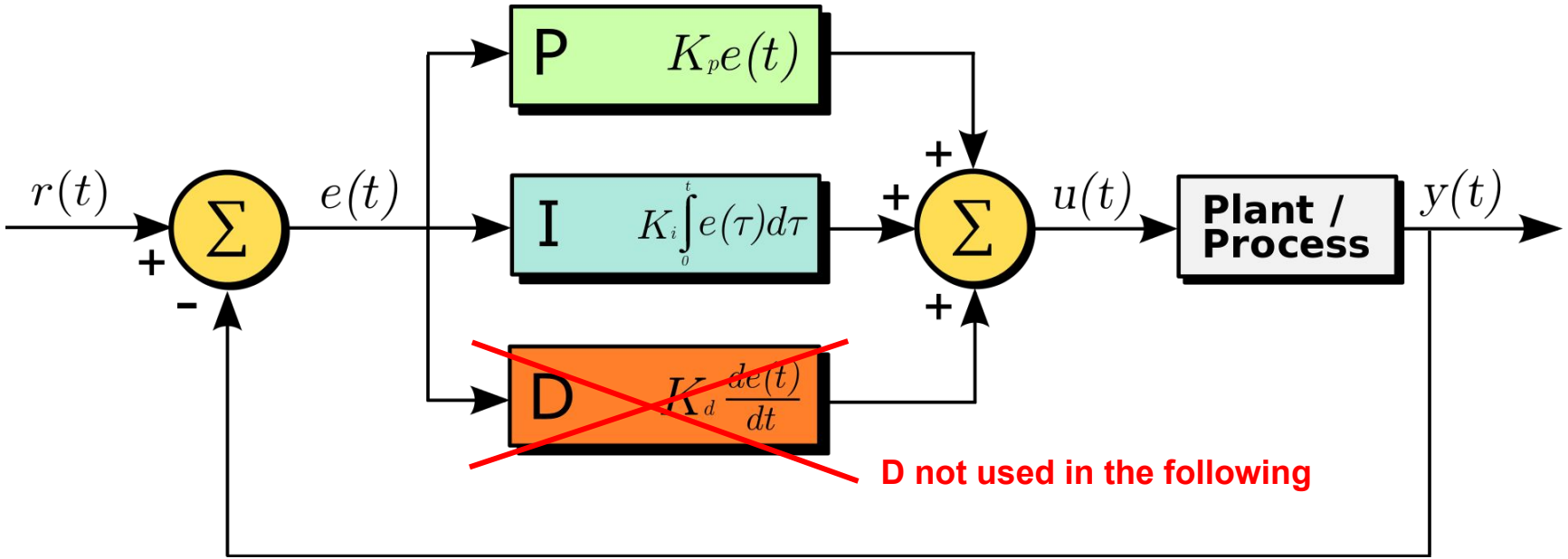
**Subtraction example**

25−8 = 25+(−8) = 17:

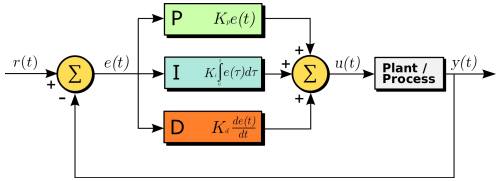
```
0001 1001   // 25
1111 1000   // −8
0001 0001   // 17
```

**Careful!**
**Pad positive numbers with 0's,**
**and negative numbers with 1's**

# (3) PID control

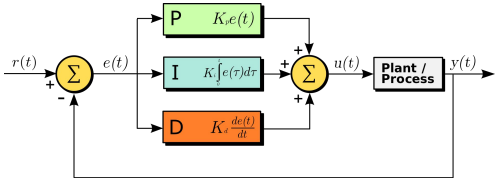


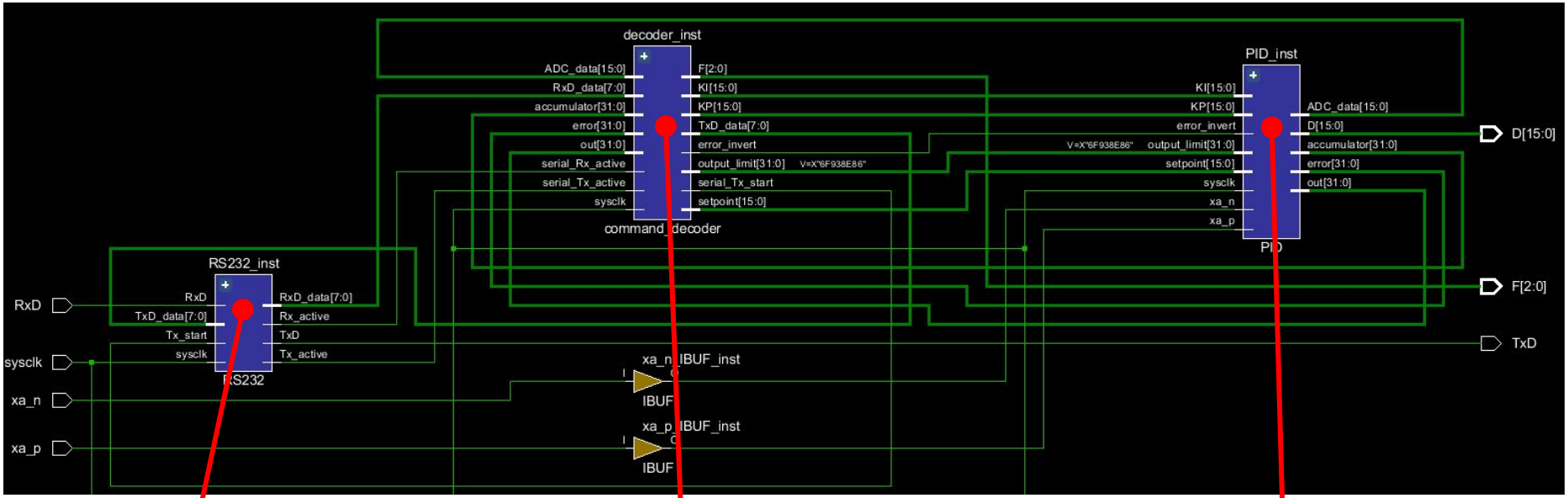
# (3.1) Control loop




```
always @(negedge(ADC_ready)) begin
   // calculate error
   error[31:0] = setpoint[15:0] - ADC_data[15:0];
   accumulator[31:0] = accumulator[31:0] + error[31:0];

   // calculate PID control output
   out[31:0] =  -(KP*(error[31:0]<<8) + KI*accumulator[31:0]);

   // check output is positive; else make it zero
   if (out[31] == 1)
     out = 0;
end
```

# (3.2) Entire PID controller



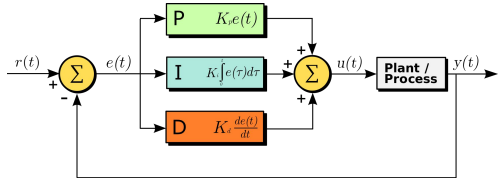



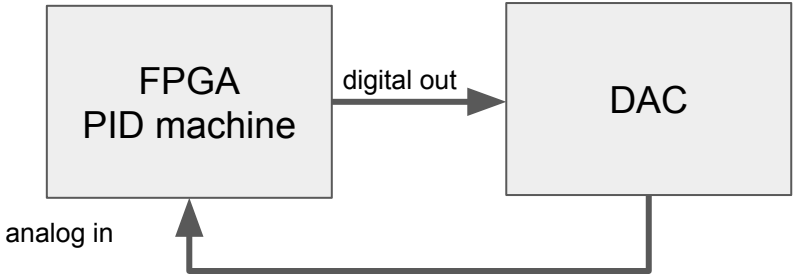

RS232 core

decodes RS232 commands into setpoint, $K_P$, $K_I$, etc.

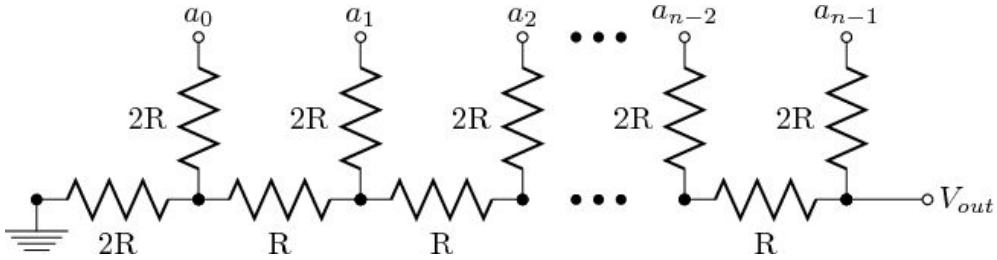
PID loop

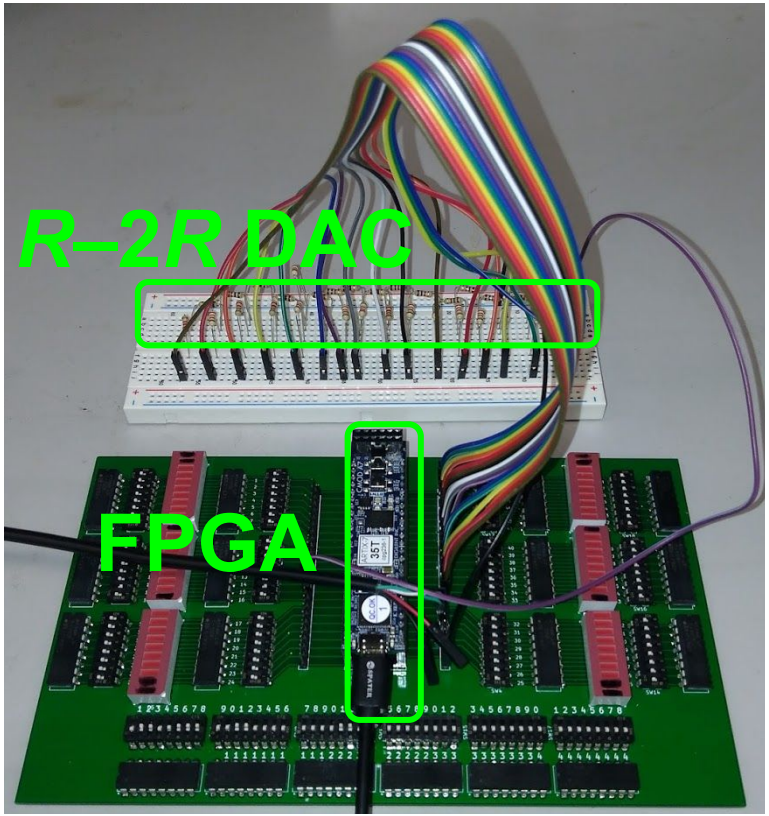
# (3.3) Testing the controller



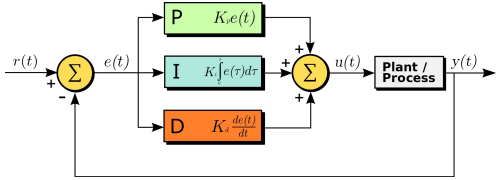



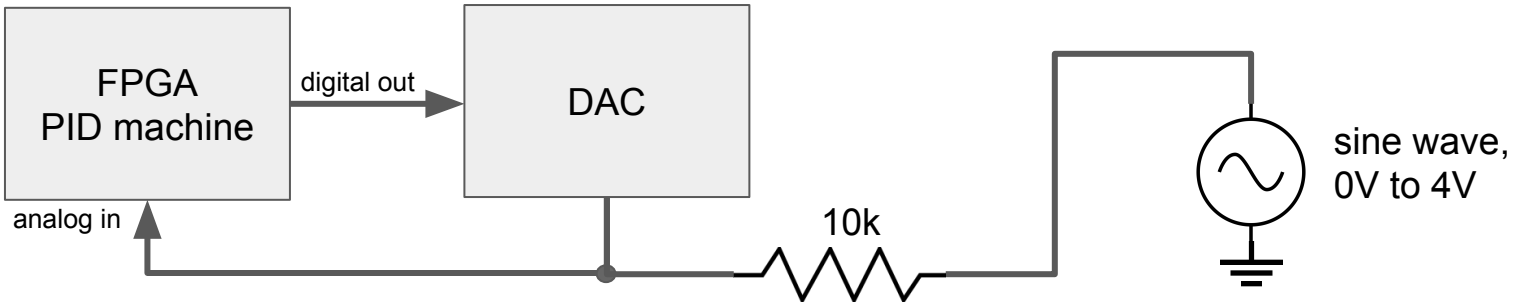

*R*–2*R* digital-to-analog converter:

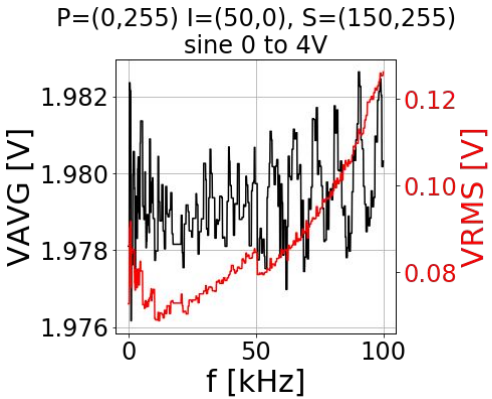


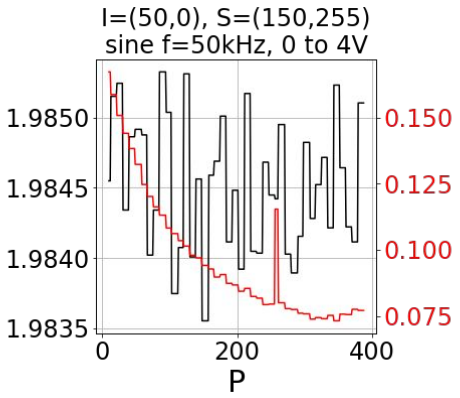



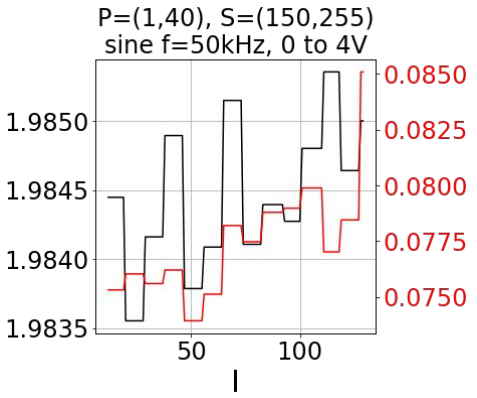
# (3.4) With sine perturbation



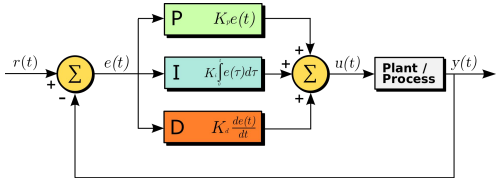



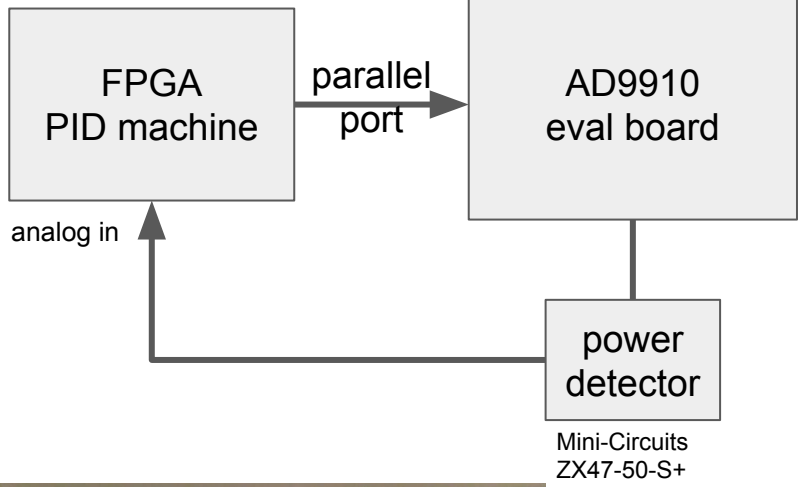



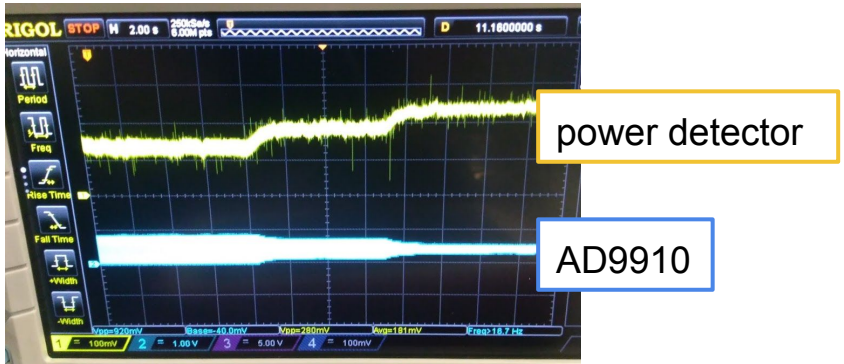



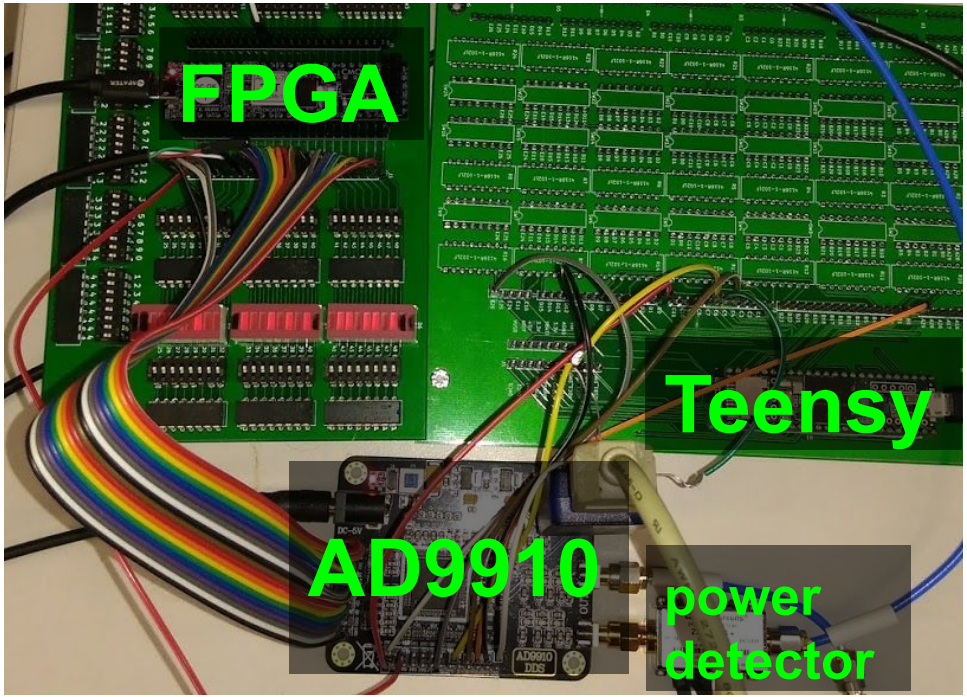



# (3.5) PID controlling AD9910

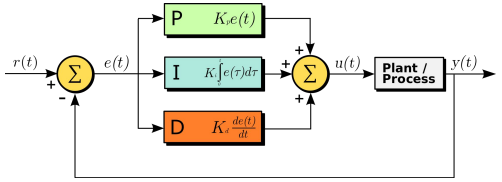


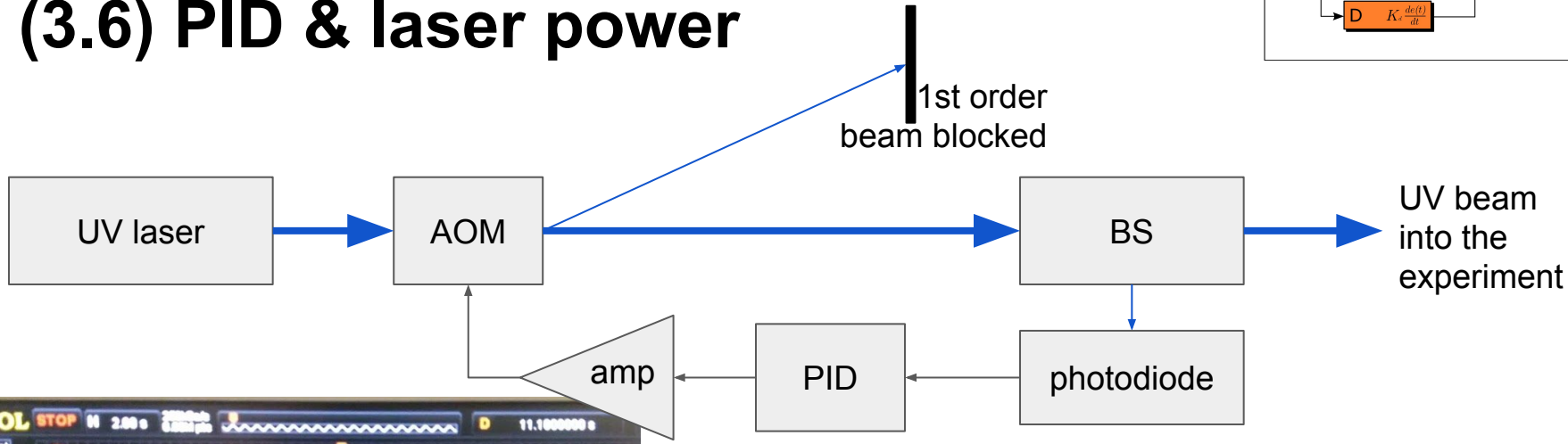



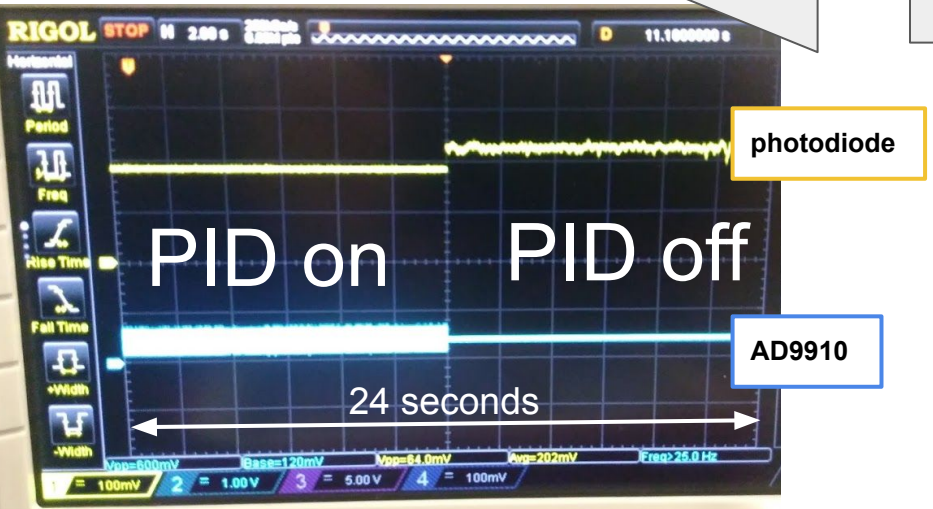



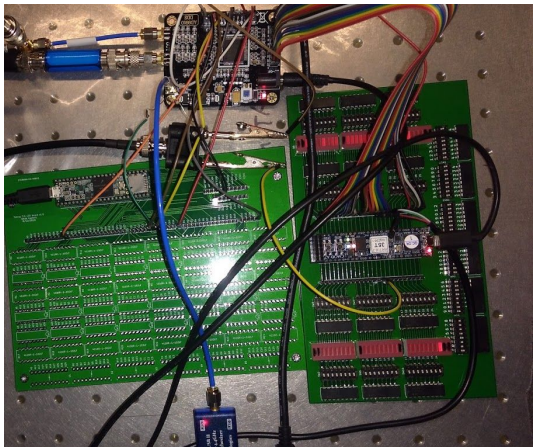



# (3.6) PID & laser power

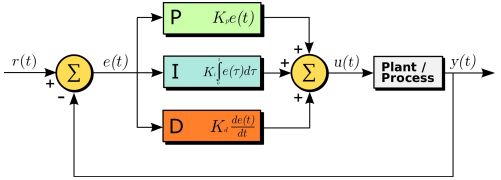


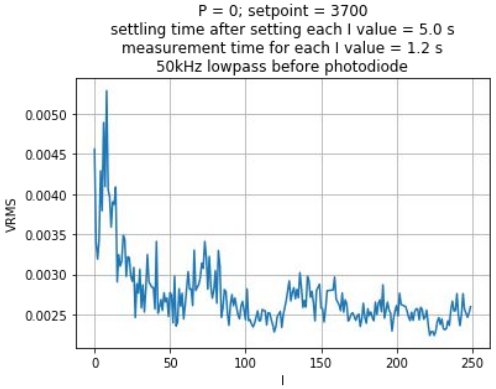



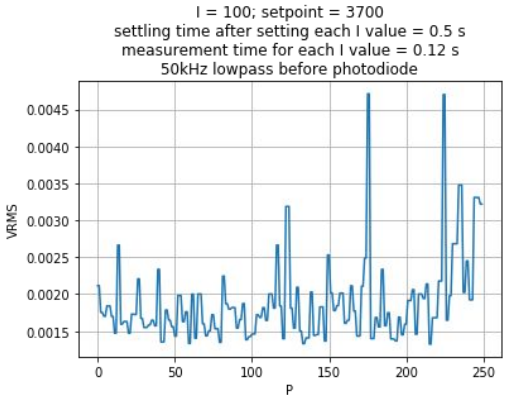



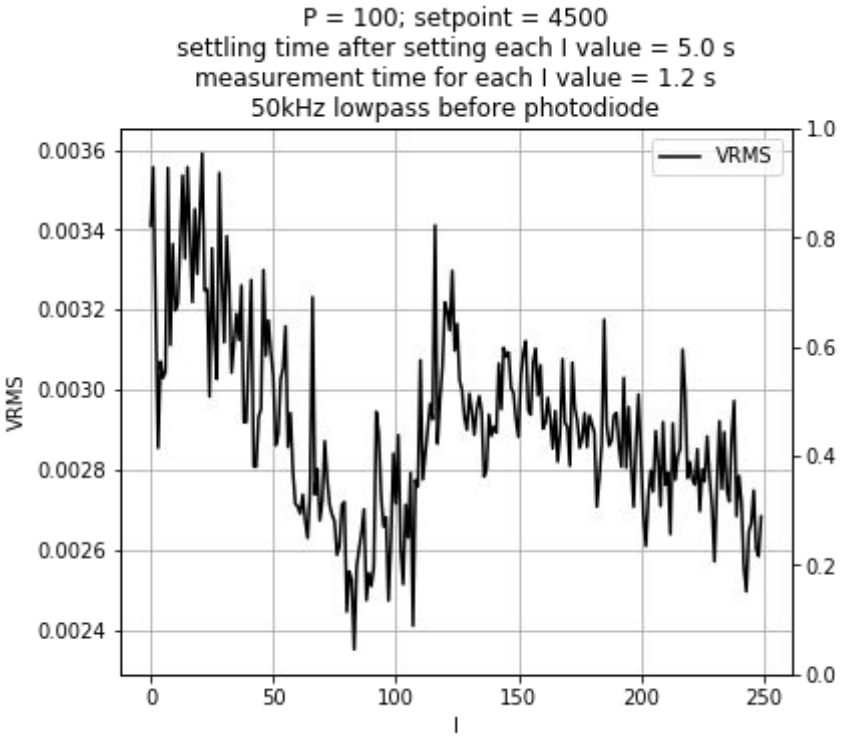



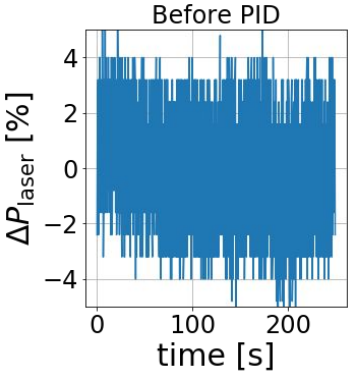
# (3.7) Residual noise

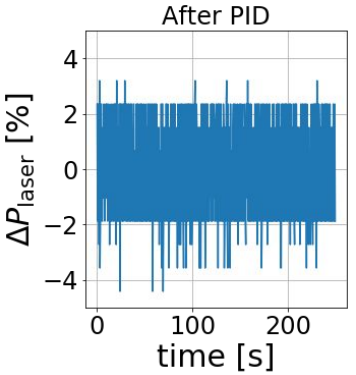


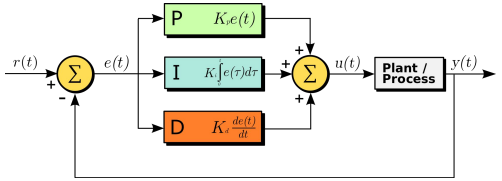



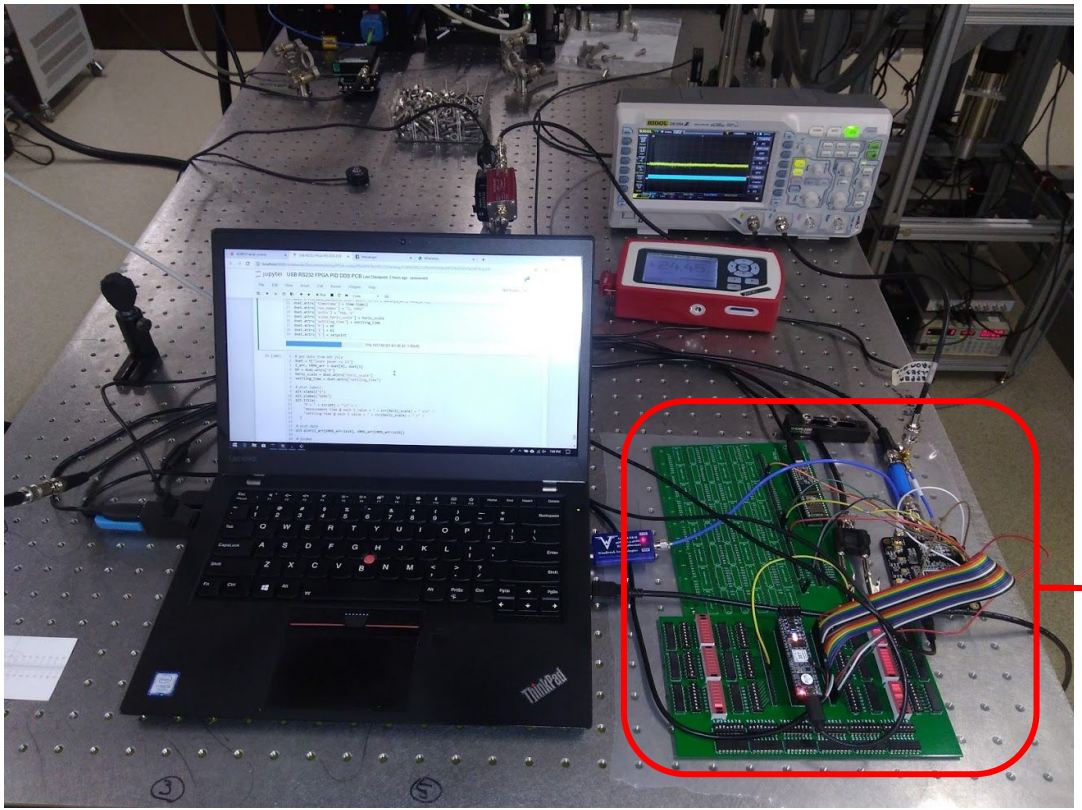



# (3.8) TODO



- replace current setup with single PCB (when it arrives)
- measure noise spectrum after locking & bandwidth