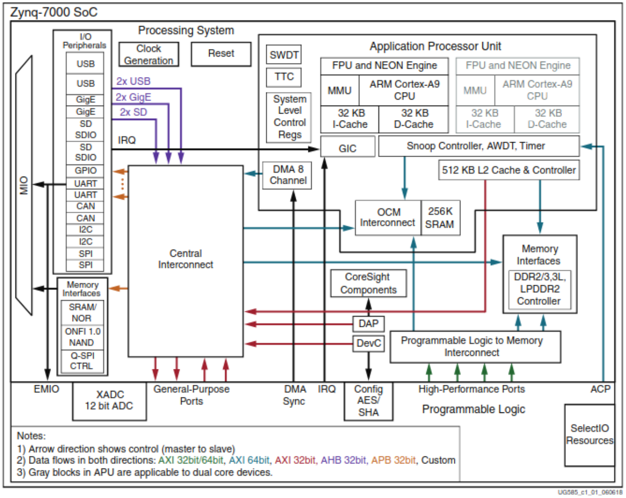
PWM Controller with Zynq SoC

Antonio Minighin

Jen 13, 2023

Block Diagram



Notes:
1) Arrow direction shows control (master to slave)
2) Data flows in both directions: AXI 32bit/64bit, AXI 64bit, AXI 32bit, AHB 32bit, APB 32bit, Custom
3) Gray blocks in APU are applicable to dual core devices.

Main Features

Processing System

The PS is made of an APU with ARM Cortex-A9 CPU featuring dual-issue, partially out of order pipeline with DSP extension, FPU, and NEON Engine for high performance, power optimized signal processing.

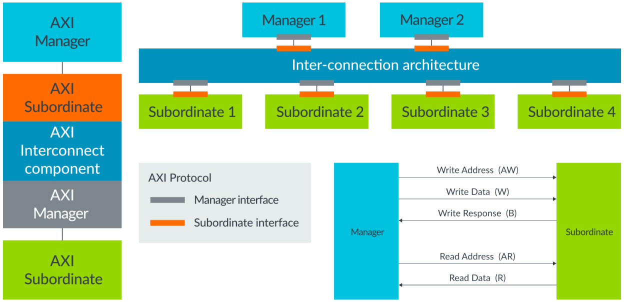
Programmable Logic

The PL is based on Artix-7 fabric that interconnects multiple components such as:
CLB with LUT, shift registers and adders
BRAM of 36Kb with dual port and FIFO logic
DSP48E1 blocks with pre-adder, multiplier and ALU
XADC with two 12 bits 1 Msample/s synchronous channels

AXI PS-PL Ports

AXI ports are used to transfer data from the PS to the PL and viceversa.
Main AXI Ports:
-General Purpose: 2 Managers and 2 Subordinates connected to the central interconnect
-High Performance: 4 Subordinates directly connected to main memory controllers
-Accelerated Coherent: one Subordinate connected to the snoop controller and the cache

Advance eXtensible Interface



AXI Protocols

There are 3 axi protocols which support different functionalities:

- AXI is memory mapped and can perform burst transactions
- AXI-Lite is a simpler subset of AXI
- AXI-Stream is a unidirectional fast transfer protocol

AXI Stream Protocol

Signals

AXIS protocol has a relative small ammount of signals:
- ACLK All signals are sampled on the rising edge
- ARESETn Active low reset signal
- TDATA: Streams data across the interface
- TVALID: Indicates that the master is driving a valid transfer
- TREADY: Indicates that the slave can accept a data transfer
- TLAST: Indicates the end of a packet of data
- TID: Indicates the source of the streamed data

Handshake

AXIS protocol allows components to exchange data using an handshake procedure.
The Manager controls **TDATA** and **TVALID** signals while the subordinate the controls the **TREADY** signal.
Both the Manager and the Subordinate controls the stream rate using simple rules:
- PLAN: The Manager can't wait the Subordinate **TREADY** befor asserting **TVALID**
- KEEP: Once the Manager asserts **TVALID** it must remain asserted utill the handshake ends.
- LAZY: The Subordinate can wait **TVALID** before asserting **TREADY**
- BUSY: The Subordinate can deassert **TREADY** before **TVALID** is asserted

PWM

```
COUNTER_LOGIC : process(clk_i)
begin
  if rising_edge(clk_i) then
    if (counter_r >= unsigned(top_i)) then
      counter_r <= to_unsigned(0, counter_r'length);
    else
      counter_r <= counter_r + 1;
    end if;
  end if;
end process;
```

```
COMPARATOR_LOGIC : process(counter_r, duty_i)
begin
  if(counter_r < unsigned(duty_i)) then
    pwm_o <= '1';
  else
    pwm_o <= '0';
  end if;
end process;
```

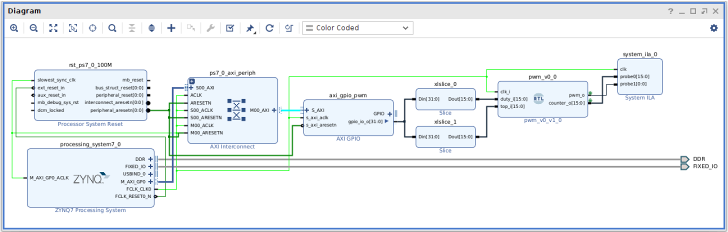
Counter (Sequential)

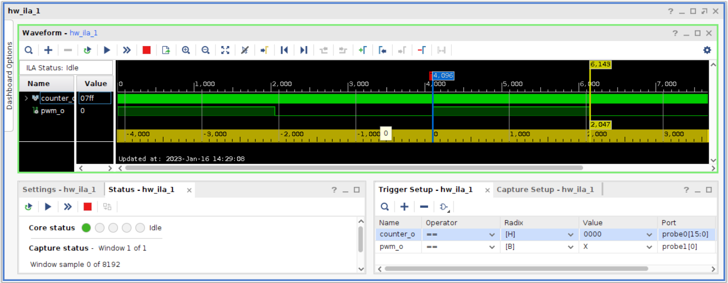
Every clock pulse the counter checks if the top value is reached choosing to wrap around zero or increment its value.

Comparator (Combinational)

The counter is continuously compared with the dutycycle. If it overcomes the dutycycle value the pwm output is pulled low

PWM test setup





ILA Status: Idle

Name	Value
counter_o	07ff
pwm_o	0

Updated at: 2023-Jan-16 14:29:08

Core status Idle

Capture status - Window 1 of 1

Window sample 0 of 8192

Name	Operator	Radix	Value	Port
counter_o	==	[H]	0000	probe0[15:0]
pwm_o	==	[B]	X	probe1[0]

PWM features

System parameters

- $F_{clk} = 100MHz$ is the master clock frquency
- $N_{bit} = 16$ is the number of bit for all the main registers
- $counter, top, duty \in [0 : 2^{N_{bit}} - 1]$

$$F_p = \frac{F_{clk}}{top + 1} > F_{p,min} = \frac{F_{clk}}{2^{N_{bit}}} \tag{1}$$

Max resolution of 16 bits at 100 MHz clock

The maximum resolution is get at the minimum frequency period, with these specifications $F_{p,min} \approx 1.5kHz$

Equivalent Number Of Bits

The duty-cycle can be modulated with a set of values limited by the counter top value, therefore an effective (or equivalent) number of bits can be defined

$$ENOB = \log_2[top + 1] = \log_2 \left[\frac{F_{clk}}{F_p} \right] \tag{2}$$

$ENOB$	8	10	12	14	16
$F_p\,[kHz]$	390	97	24	6.1	1.5
$T_p\,[\mu s]$	2.56	10.24	40.96	163.84	655.36

XADC

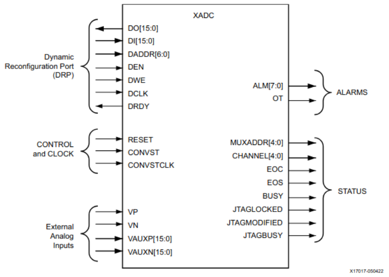
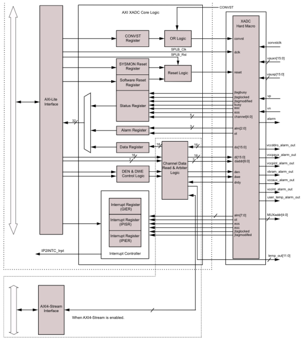
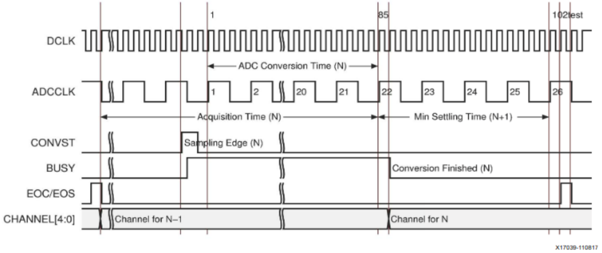


Figure 1-3: **XADC Primitive Ports**



XADC interface

- JTAG interface connected with PS (SLOW)
- AXI wrapper interface in the PL (FAST)

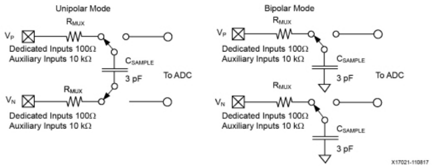


XADC Signals

- DLCK is the clock over which the DRP communicates
- ADCCLK is locked to DCLK/4
- CONVST edge starts the conversion at the next ADCLK
- EOC/EOS signal when the data is stored into the register

The converstion lasts 21 ADCCLK periods. The data takes other 16 DCLK periods to be stored in the register.

Acquisition Timing



Settling time resolution

N_{bit} resolution requires a relative error

$$e^{-\frac{t}{RC}} \leq \frac{1}{2^{N_{bit}+1}}$$

which requires a settling time of

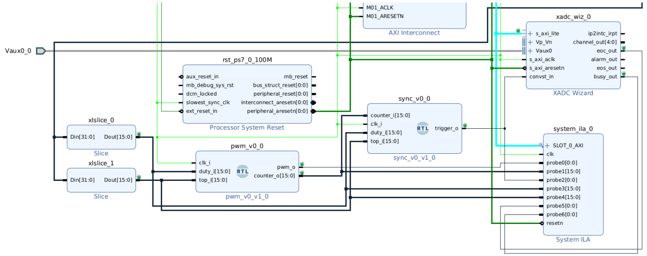
$$t_{settle} \geq RC \ln(2^{N_{bit}+1})$$

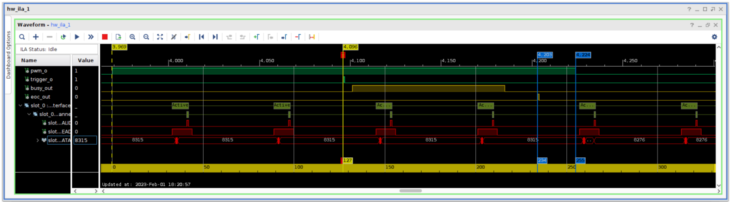
$$t_{ACQ} = 9 \times (R_{MUX} + R_{MUX}) \times C_{SAMPLE} \tag{3}$$

Path Mode	AUX Unipolar	AUX Bipolar	V_P/V_N Unipolar	V_P/V_N Bipolar
[1] t_{ACQ} [ns]	540	270	5.4	2.7

[1]External source impedances must be add to R_{MUX}

ADC test setup



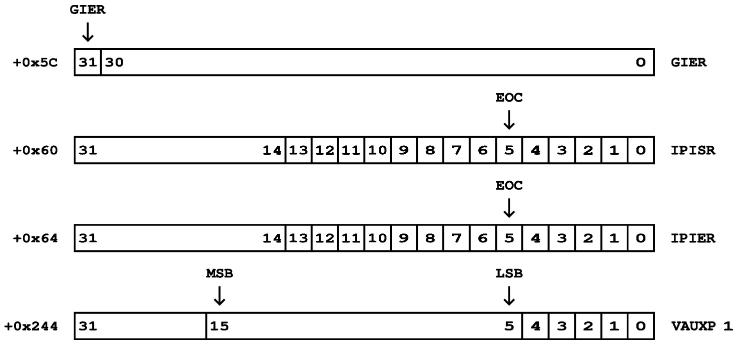


Name	Value
pwm_o	1
trigger_o	1
busy_out	0
eoc_out	0
slot_0 :...terface	_
slot_0...anne	_
slot...ALID	0
slot...EADY	0
slot...ATA	8315

AXI XADC interrupts

Registers

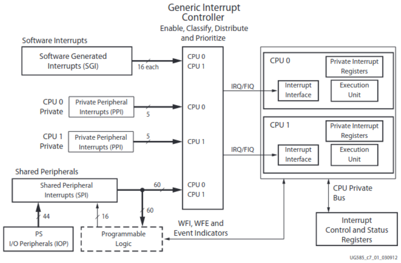
- GIER: Enable the `ip2intc_irpt` line to signal an IRQ
- IPIER: Enable individual events to control the `ip2intc_irpt` line
- IPISR: When the events occours, the corresponding bit is activated



CPU interrupts

Snoop Control Unit

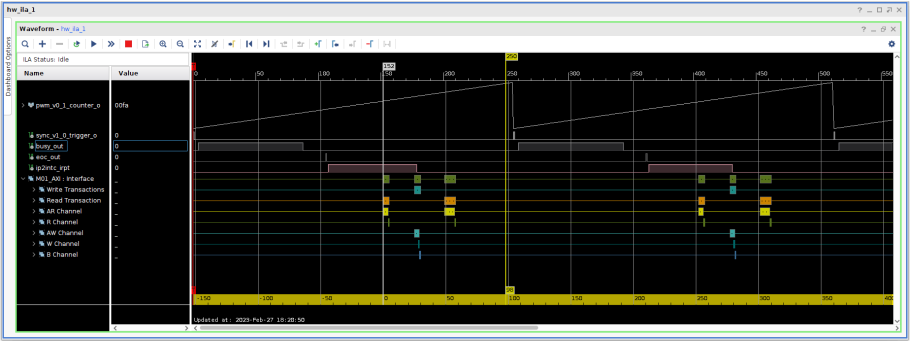
The ARM Cortex-A9 architecture supports multiple CPUs which may have access to the same set of resources. To ensure coherency between the CPU operations a SCU is paired to them. The Interupt Controller and the associated registers are embedded here.


Generic Interrupt Controller
Enable, Classify, Distribute and Prioritize
Software Interrupts
Software Generated Interrupts (SGI)
16 each
CPU 0
CPU 1
CPU 0 Private
Private Peripheral Interrupts (PPI)
5
CPU 1 Private
Private Peripheral Interrupts (PPI)
5
Shared Peripherals
Shared Peripheral Interrupts (SPI)
60
44
16
60
PS I/O Peripherals (IOP)
Programmable Logic
WFI, WFE and Event Indicators
IRQ/FIQ
IRQ/FIQ
CPU 0
Private Interrupt Registers
Interrupt Interface
Execution Unit
CPU 1
Private Interrupt Registers
Interrupt Interface
Execution Unit
CPU Private Bus
Interrupt Control and Status Registers
UG585_c7_01_030912

Generic Interrupt Controller

The GIC is a non-vectored interrupt controller, therefore a unique master handler is associated to each CPU. This handler polls the GIC to get the ID of the device that requested the interrupt.

```
//Associate the SCU GIC handler to the IRQ Exception
Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_IRQ_INT,
               (Xil_ExceptionHandler)XScuGic_InterruptHandler, &InterruptController);
//Enable the Exception
Xil_ExceptionEnable();
//Connect a handler to the device interrupt ID
XScuGic_Connect(&InterruptController, INTC_DEVICE_INT_ID,
               (Xil_ExceptionHandler)DeviceDriverHandler, (void *)&InterruptController);
//Enable the device interrupt ID
XScuGic_Enable(&InterruptController, INTC_DEVICE_INT_ID);
```



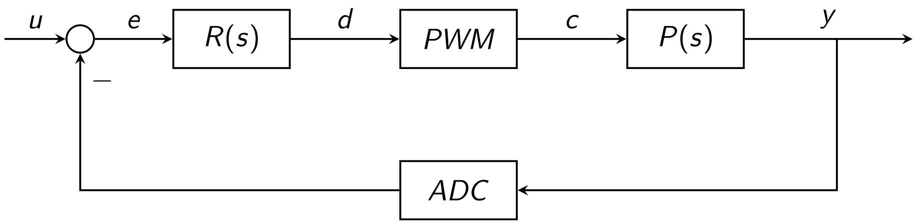
CPU Interrupt Handler

Handler Limits ($T_{clk} = 100 MHz$)

The CPU enters the handler with a huge (random) delay $\approx 50\,T_{clk}$ and must perform a write transaction to clear the interrupt. Without further improvements, interrupts faster than $\approx 250\,T_{clk}$ may be missed. This limits the acquisition rate to 400 $Ksample/s$

```
void DeviceDriverHandler(void *CallBackRef){
        // Read the Interrupt Status Register should be done
        // to check which event has called the interrupt
        u32 reg = XADC_mRead(AXI_XADC_BASE_ADDRESS, IPISR_OFFSET);
        XADC_mWrite(AXI_XADC_BASE_ADDRESS, IPISR_OFFSET, reg); // The IPISR has toggle behaviour
        adc = XADC_mRead(AXI_XADC_BASE_ADDRESS, VAUX1_OFFSET);
        // THE ALGHORITM FOLLOWS //
}
```

Loop Control



$$PWM = \frac{V_{cc}}{TOP + 1} \qquad ADC = \frac{2^{N_{bit}}}{FSR}$$

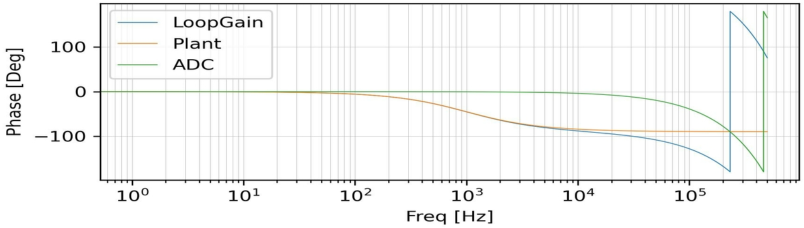
PWM Mean

The PWM scale factor is valid only for it's mean value.

ADC+CPU Delay

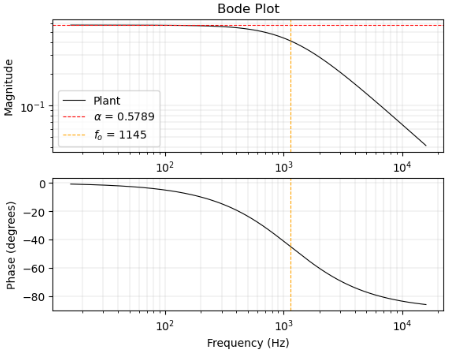
The ADC and the CPU take some time to convert and process the data introducing a delay $e^{-s\Gamma}$ that degrades the phase margin

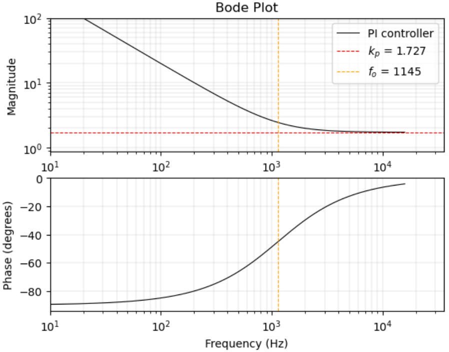
Sample and Hold $e^{-sT/2}$



F/f	2	4	5	8	10	15	20	30	60	100
$\angle e^{-j\omega T/2}$	-90	-45	-36	-22.5	-18	-12	-9	-6	-3	-1.8

Plant and Controller





RC network

$$P(s) = \frac{\alpha}{1+s\tau_o}$$
$$\tau_o = C(R_s//R_p)$$
$$\alpha = \frac{R_p}{R_s+R_p}$$

$$R_s = 2.4\ k\Omega$$
$$R_p = 3.3\ k\Omega$$
$$C = 100\ nF$$

PI controller

$$R(s) = k_p + \frac{k_i}{s}$$
$$k_p\alpha = 1$$
$$k_i = \omega_o k_p$$

$$k_p = 1.727$$
$$k_i = 12429\ s^{-1}$$

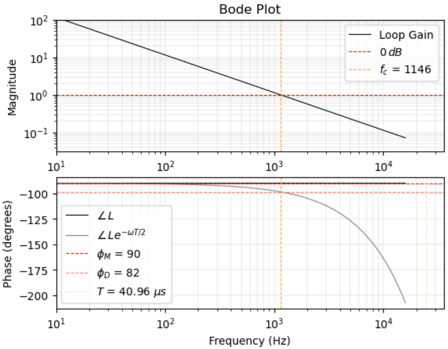
Loop Gain and Phase margin

ADC parameters

$FSR = 3.3\ V$
$Nbit = 12$
$\beta = \frac{2^{12}}{3.3}\ V^{-1}$

PWM parameters

$V_{cc} = 3.3\ V$
$ENOB = 12$
$\alpha_m = \frac{3.3}{2^{12}}\ V$



Loop Gain

$L(s) = \frac{\alpha k_p}{1+s\tau_o} \frac{s+k_i/k_p}{s}$
if $\frac{k_i}{k_p} = \omega_o$ then $L(s) = \frac{\alpha k_p}{s\tau_o}$

Closed Loop Gain

$A_F(s) = \frac{1}{\beta} \frac{L}{1+L}$
which for $L >> 1 \rightarrow \frac{1}{\beta}$
but for $L << 1 \rightarrow A$

Discretization

Backward Euler ($s = \frac{1-z^{-1}}{T}$)

From the fake-linearization of $z = \frac{1}{e^{-sT}} \approx \frac{1}{1-sT}$

$$\frac{df(t)}{dt}\Big|_{t=kT} = \frac{f(kT) - f(kT-T)}{T}$$

Tustin Method ($s = \frac{2}{T}\frac{z-1}{z+1}$)

From the fake-linearization of $z = \frac{e^{sT/2}}{e^{-sT/2}} \approx \frac{1+sT/2}{1-sT/2}$

$$\frac{f'(kT+T) + f'(kT)}{2} = \frac{f(kT+T) - f(kT)}{T}$$

Frequency Response

Frequency Warping

For $f << F$ the Euler discretization keeps the frequency response

$$s = \frac{1 - z^{-1}}{T} = \frac{1 - e^{-j\omega T}}{T} \approx j\omega$$

Tustin discretization introduces a much severe distortion

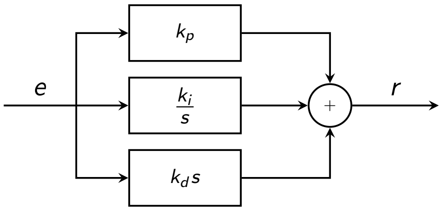
$$s = \frac{2}{T}\frac{z-1}{z+1} = \frac{2}{T}\frac{e^{j\omega T}-1}{e^{j\omega T}+1} = \frac{2}{T} j \tan(\pi f T)$$

Pre-warping

The design procedure must include a pre-warping operation to get the target frequency response $\overline{f}$ according to the law

$$f = \frac{\arctan(\pi T \overline{f})}{\pi T}$$

Discrete PID



$PID = k_p + \frac{k_i}{s} + k_d s$

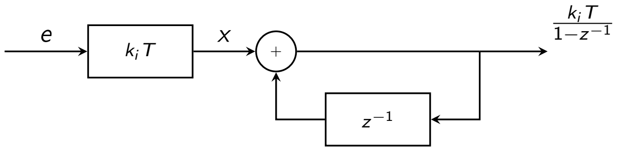
Euler discretization:

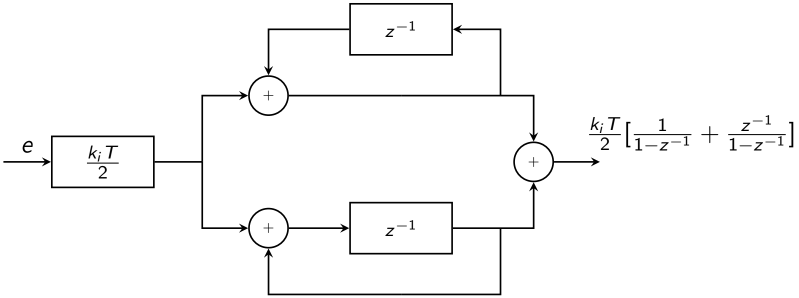
$$PID(z) = k_p + \frac{k_i T}{1 - z^{-1}} + k_d \frac{1 - z^{-1}}{T}$$

Tustin discretization:

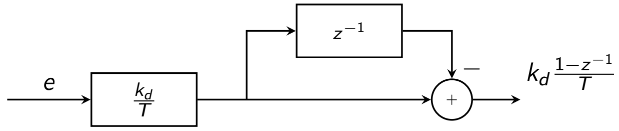
$$PID(z) = k_p + \frac{k_i T}{2} \frac{1 + z^{-1}}{1 - z^{-1}} + k_d \frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}}$$

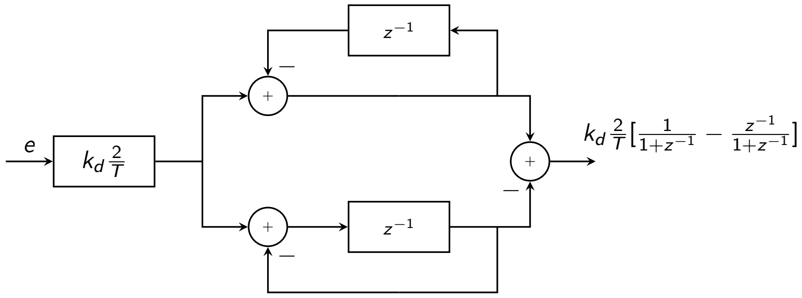
Integrator k_i/s




e
$\frac{k_i T}{2}$
$+$
z^{-1}
$+$
z^{-1}
$+$
$\frac{k_i T}{2}\left[\frac{1}{1-z^{-1}} + \frac{z^{-1}}{1-z^{-1}}\right]$

Differentiator $k_d\, s$




e
$k_d \frac{2}{T}$
z^{-1}
$-$
$+$
$+$
$-$
$+$
z^{-1}
$-$
$k_d \frac{2}{T}\left[\frac{1}{1+z^{-1}} - \frac{z^{-1}}{1+z^{-1}}\right]$

Oversampling

Hardware Parameters

The PWM runs at the clock frequency $F_{clk} = 100\,MHz$
Both the PWM and the ADC have $ENOB = N_{bit} = 12$
The SYNC triggers the ADC at fractions of $T_p = 40.96\,\mu s$

OSR	1	2	4	8
$F\,[kHz]$	24	48	97	195
$T\,[\mu s]$	40.96	20.48	10.24	5.12
$k_i = 12429$				
$k_i\,T$	0.5092	0.2546	0.1273	0.0636

Fixed point limits

Q codec

Rational numbers must be coded into binary form with N bits for the integer part and M for the fractional one.
With N+M bits it is possible to represent unsigned number in the range $[0 : 2^N - 1 + \sum_{m=1}^{M} 2^{-m}]$ and signed numbers in the range $[2^{N-1} - \sum_{m=1}^{M} 2^{-m} : 2^N - 1 + \sum_{m=1}^{M} 2^{-m}]$.

Oversampled Coefficients

Huge oversampling ratios may produce discrepancies on coefficients whether they are multiplied or divided by the sampling period T.
This may become a problem when coefficients iteract with other quantities because the point must always be adjusted before or after each operation.
For example, adding two 32 bit unsigned numbers that represent an integer and a Q16 number may require truncation or a 48 bit operation.

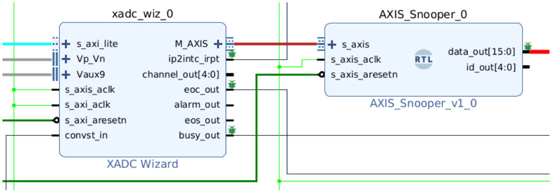
Software Algorithm

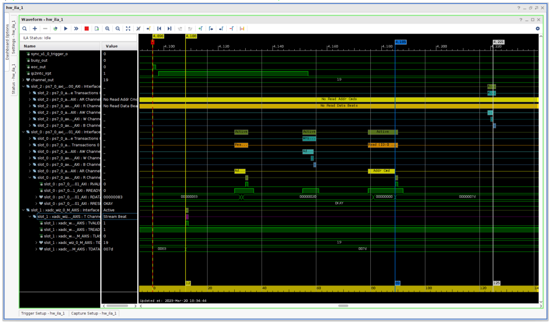
```
static uint16_t adc = 0;
static uint16_t ref = 0x777;
static int16_t err = 0;
static uint16_t kp = 221; //1.727x2^7
static uint16_t kiT = 33371; //0.5092x2^16
static int32_t x = 0; //integrator input
static int32_t acc = 0; //integrator output
static int32_t y = 0; // controller output

void DeviceDriverHandler(void *CallBackRef){
        u32 reg = XADC_mRead(AXI_XADC, IPISR);
        XADC_mWrite(AXI_XADC, IPISR, reg);

        adc = XADC_mRead(AXI_XADC, VAUX) >> 4;

        err = (int16_t) (ref - adc);
        x = (kiT*err)>>kiShift;
        acc = acc + x;

        y = acc + ((kp*err) >> kpShift);
        if (y < y_min) {
                y = y_min;
        }
        else if (y > y_max) {
                y = y_max;
        }

        duty = (uint32_t) y;
        AXIREG_mWriteReg(AXI_REG, AXI_DUTY, duty);
}
```

Numeric Representation

The first step is to calculate the error from the difference of two positive values. This two values are uint16_t that can represent numbers in the range $[0 : 2^{16} - 1]$, however the difference may be in the range $[-2^{16} + 1 : 2^{16} - 1]$ which should require a signed integer of 17 bits. The second step is to multiply a int16_t with a uint16_t that can produce a number in the range $[-2^{31} + 2^{15} : 2^{31} - 2^{16} - 2^{15} - 1]$ which requires a 32 bit signed accumulator.

Stream and Catch

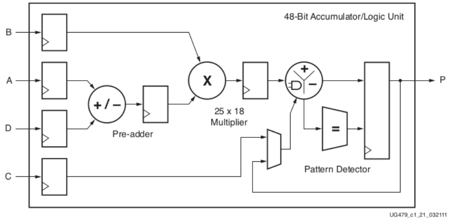




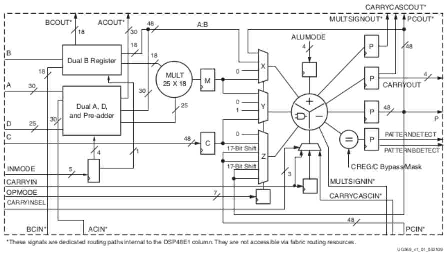
AXIS Snooper

In a real time acquisition system the ADC continuously samples and streams the data but the AXI stream bus data rate is governed by handshakes and may not be constant. To have a constant stream the Manager and the Subordinate must agree on the streaming method. One way is to let the Subordinate be always ready, this forces the streaming but it's not reliable. The second way is to use the LAZY property and let the Subordinate be ready every time the Manager validates a transaction. The Snooper also checks the **TID** of a data packet, keeping those of interests and passing through the others.

DSP Block



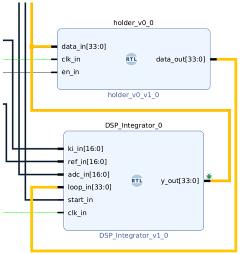
```
PIPELINE : process(clk_in)
begin
    if rising_edge(clk_in) then
        if (en_in = '1') then
            a <= signed(a_in);
            b <= signed(b_in);
            c <= signed(c_in);
            d <= signed(d_in);
            sub <= a - d;
            mult <= sub*b;
            p <= mult + c;
        end if;
    end if;
end process;
```



Description

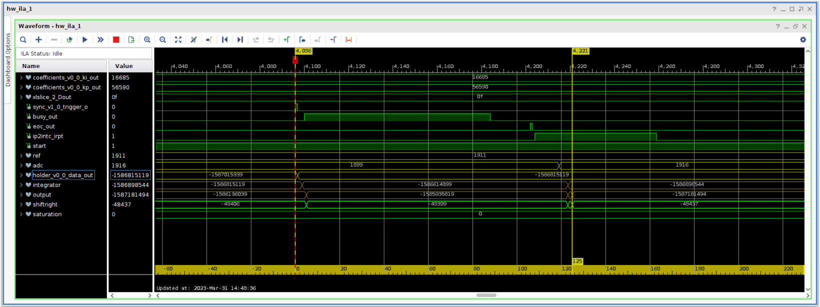
DSP Blocks are hard cores used for several purpose, they embedd a 30-25 bits Pre-adder, a 25x18 bits Multiplier, a 48 bits ALU as well as other more complex functionalities.

DSP Integrator



Integrating Loop

An integrator can be created by looping back the output to the ALU input.
To reduce the operating frequency a sampling register is activate once per period holding the output value in the feadback path.



ILA Status: Idle

Name	Value
coefficients_v0_0_ki_out	16685
coefficients_v0_0_kp_out	56590
xlslice_2_Dout	0f
sync_v1_0_trigger_o	0
busy_out	0
eoc_out	0
ip2intc_irpt	1
start	1
ref	1911
adc	1916
holder_v0_0_data_out	-1586815119
integrator	-1586898544
output	-1587181494
shiftright	-48437
saturation	0

```
Updated at: 2023-Mar-31 14:48:36
```

PI Implementation

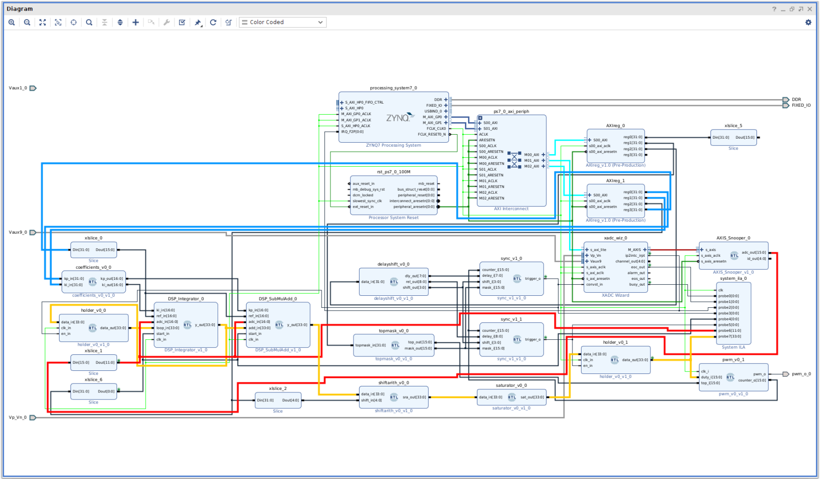


Diagram
Color Coded
processing_system7_0
ZYNQ7 Processing System
ps7_0_axi_periph
AXI Interconnect
rst_ps7_0_100M
Processor System Reset
AXIreg_0
AXIreg_1
AXIreg_v1.0 (Pre-Production)
xlslice_5
xadc_wiz_0
XADC Wizard
AXIS_Snooper_0
AXIS_Snooper_v1_0
system_ila_0
System ILA
xlslice_0
coefficients_v0_0
holder_v0_0
xlslice_1
xlslice_6
DSP_Integrator_0
DSP_SubMulAdd_0
delayshift_v0_0
sync_v1_0
sync_v1_1
topmask_v0_0
holder_v0_1
pwm_v0_1
xlslice_2
shiftarith_v0_0
saturator_v0_0
Vaux1_0
Vaux9_0
Vp_Vn_0
DDR
FIXED_IO
pwm_o_0