System on Chip Design Lab Report

M.Sc. Embedded Systems Design

Lab: - **AXI-Stream IP: Hardware Accelerated DSP (PI Controller)**

Under the Guidance of

Prof. Dr.-Ing. Kai Müller

Submitted by

Joe Joseph Nelson

38750

Table of Contents

1 List of figures

2 Objective

AXI-Stream IP: Hardware Accelerated DSP (PI Controller)

A PI controller shall be designed as AXI-S (AXI-Stream) IP for a System-on-Chip design.

A set of template files are provided in the Download section.

The AXI stream interface is provided and does not require modifications.

The lab is mandatory and requires a written report.

Critical verification by simulation must be part of the project.

3 Design

The Proportional Integral (PI) controller is one of the commonly used controller. It is a control loop mechanism employing feedback that is widely used in industrial control systems and a variety of other applications requiring continuously modulated control. It gives proper ramp up to achieve desired value quickly as well as the occurrence of offset error or steady state error about desired value has been eliminated by the integral term.

The block diagram of the PI controller implemented is shown in the figure below.

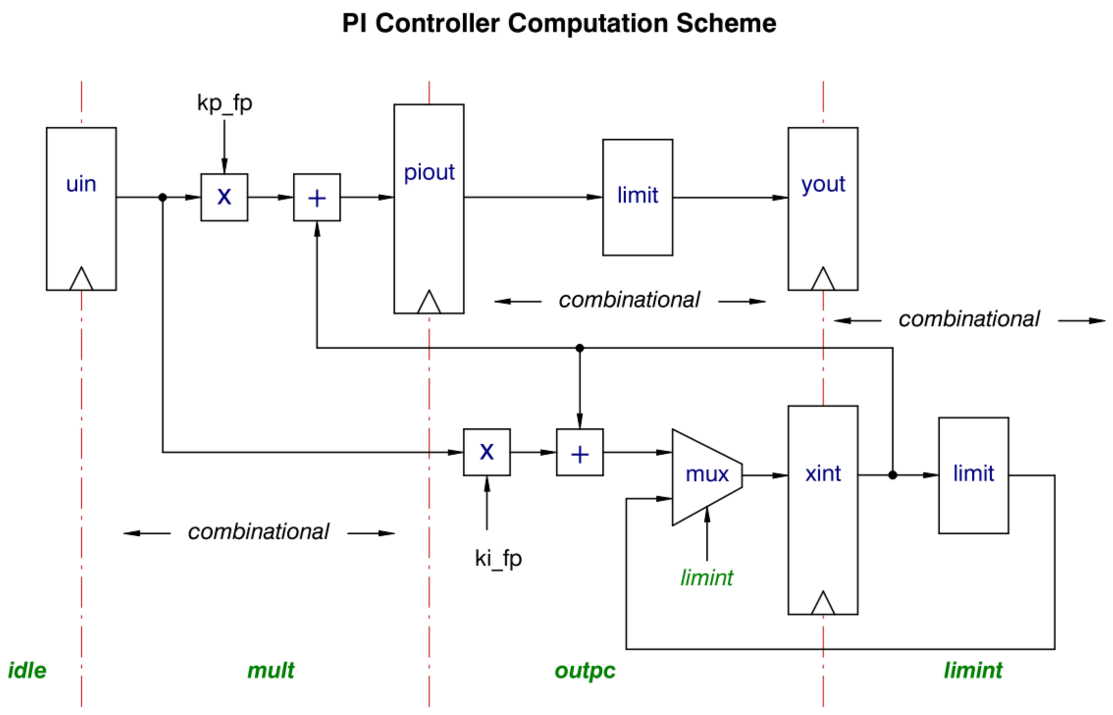


Figure 1 Block Diagram of PI controller

The structure needs to be implemented in the VHDL code as state machine. The states that are required and the corresponding functions that need to be completed, are clearly shown in the diagram. Uin is the input to the system, ki_fp (integral gain) and kp_fp (proportional gain) are the corresponding gains of the system. There are limiters present in the system to avoid overflow of the values. Each of the states are responsible for introducing the unit delay in the system.

The functionality is implemented in an AXI-S slave module, the input to this is provided by an AXI-S master connected to it. The PI controller can be coded using a simple state machine with 4 states. The state diagram is as shown in the below figure. Each of the states and its functionality can be understood from the above block diagram.

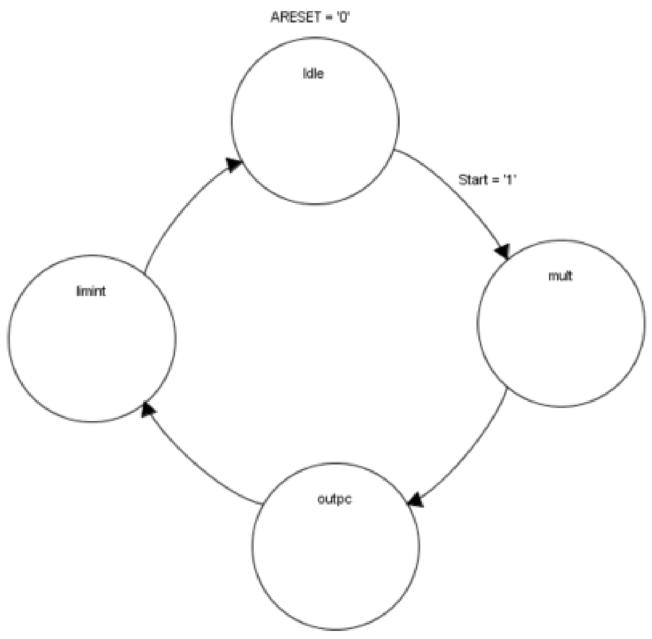
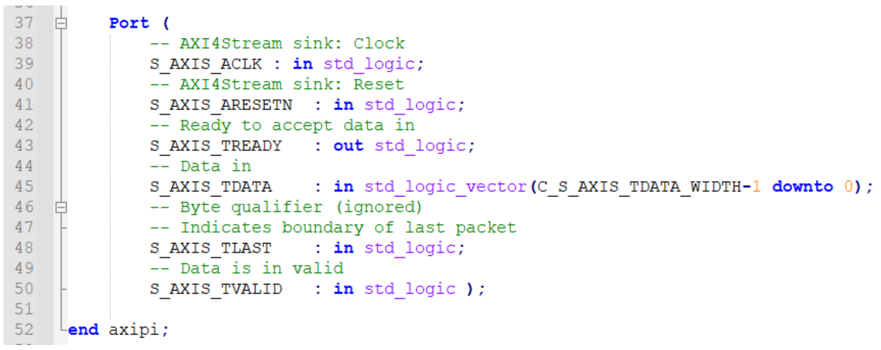


Figure 2 State diagram of PI controller

As shown in figure 2, we need 4 states to implement the PI controller. The state machine is in idle state when the reset is removed. It starts the PI controller action if there is a data transmission from the master indicated by the start signal. When start is one, it moves to the mult state where the multiplication of proportional gain with input happens. After mult state it moves to outpc, where controller output and integral gain is calculated. State machine then moves to limint state where the xint to be stored is limited to avoid overflowing. This summarizes the function of PI controller state machine.

4 Source Code

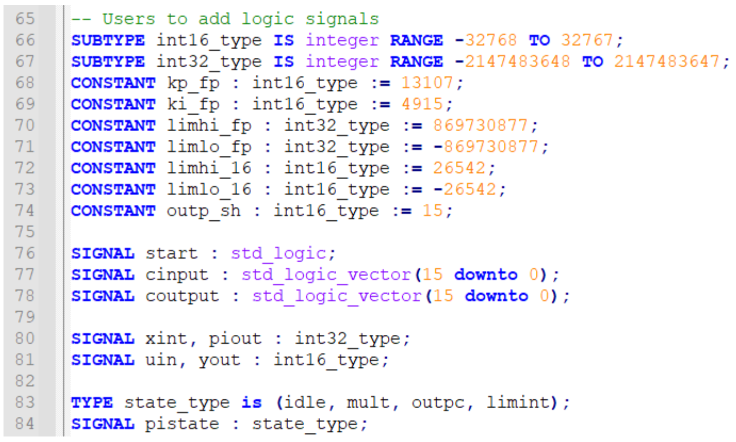
The PI controller is implemented inside an AXI-S slave module. The interfaces of the module are AXI-S signals only.



```vhdl
    Port (
        -- AXI4Stream sink: Clock
        S_AXIS_ACLK : in std_logic;
        -- AXI4Stream sink: Reset
        S_AXIS_ARESETN  : in std_logic;
        -- Ready to accept data in
        S_AXIS_TREADY   : out std_logic;
        -- Data in
        S_AXIS_TDATA    : in std_logic_vector(C_S_AXIS_TDATA_WIDTH-1 downto 0);
        -- Byte qualifier (ignored)
        -- Indicates boundary of last packet
        S_AXIS_TLAST    : in std_logic;
        -- Data is in valid
        S_AXIS_TVALID   : in std_logic );

end axipi;
```

Figure 3 AXI-S Interfaces of module

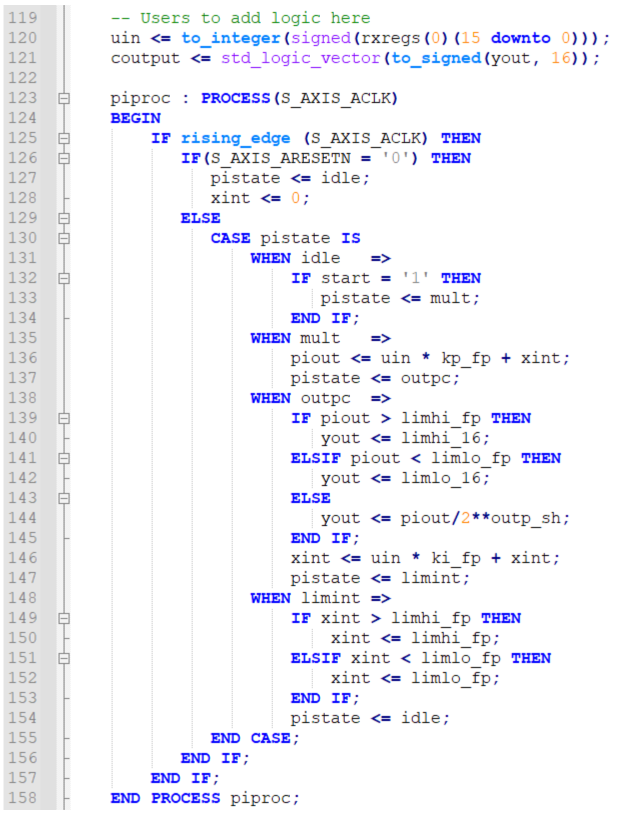
The source code explained here shows the implementation of PI controller.



```vhdl
-- Users to add logic signals
SUBTYPE int16_type IS integer RANGE -32768 TO 32767;
SUBTYPE int32_type IS integer RANGE -2147483648 TO 2147483647;
CONSTANT kp_fp : int16_type := 13107;
CONSTANT ki_fp : int16_type := 4915;
CONSTANT limhi_fp : int32_type := 869730877;
CONSTANT limlo_fp : int32_type := -869730877;
CONSTANT limhi_16 : int16_type := 26542;
CONSTANT limlo_16 : int16_type := -26542;
CONSTANT outp_sh : int16_type := 15;

SIGNAL start : std_logic;
SIGNAL cinput : std_logic_vector(15 downto 0);
SIGNAL coutput : std_logic_vector(15 downto 0);

SIGNAL xint, piout : int32_type;
SIGNAL uin, yout : int16_type;

TYPE state_type is (idle, mult, outpc, limint);
SIGNAL pistate : state_type;
```

Figure 4 Constants, Signals and State declaration for PI controller

The above code shows the declaration of constants, signals and state type required by the PI controller. The unchanged values used in the code are declared as CONSTANTS at the start of the program. ki_fp (Integral gain) 0.40 and kp_fp (proportional gain) 0.15 and the limit constants for a range of -0.81 to +0.81 to be used are declared here.



```vhdl
-- Users to add logic here
uin <= to_integer(signed(rxregs(0)(15 downto 0)));
coutput <= std_logic_vector(to_signed(yout, 16));

piproc : PROCESS(S_AXIS_ACLK)
BEGIN
    IF rising_edge (S_AXIS_ACLK) THEN
       IF(S_AXIS_ARESETN = '0') THEN
          pistate <= idle;
          xint <= 0;
       ELSE
          CASE pistate IS
              WHEN idle    =>
                  IF start = '1' THEN
                     pistate <= mult;
                  END IF;
              WHEN mult    =>
                  piout <= uin * kp_fp + xint;
                  pistate <= outpc;
              WHEN outpc   =>
                  IF piout > limhi_fp THEN
                     yout <= limhi_16;
                  ELSIF piout < limlo_fp THEN
                     yout <= limlo_16;
                  ELSE
                     yout <= piout/2**outp_sh;
                  END IF;
                  xint <= uin * ki_fp + xint;
                  pistate <= limint;
              WHEN limint =>
                  IF xint > limhi_fp THEN
                      xint <= limhi_fp;
                  ELSIF xint < limlo_fp THEN
                      xint <= limlo_fp;
                  END IF;
                  pistate <= idle;
          END CASE;
       END IF;
    END IF;
END PROCESS piproc;
```

Figure 5 Code that waits for Interrupt.

The above code shows the state machine implementation of the PI controller. The 4 states and the functional actions are shown in the code. The state machine is triggered when start signal becomes high after receiving a tlast signal from the AXI master. The function of the code is explained in the state diagram part of the document.

5 MATLAB Simulation for results comparison

The same PI controller is implemented in MATLAB and is used for the comparison of the results with the VHDL code.

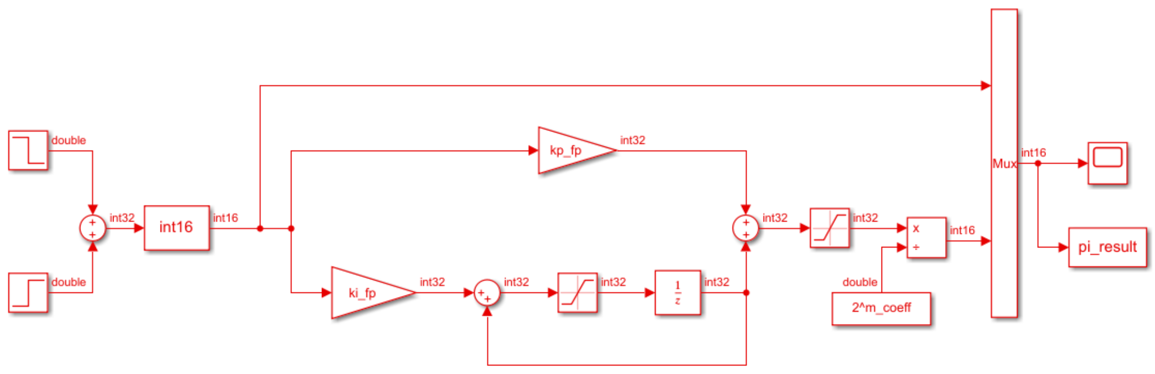


Figure 6 MATLAB implementation of the PI controller

The controller block diagram will be same as what shown for the VHDL design. The unit delay needs to be explicitly added in the MATLAB, so there is a delay unit extra in the above diagram. At the end both the input and output of the system are given to the same scope so that it can be observed in the same graph.

The comparison of input and output values of PI controller is as shown.

Input	PI out
0	0
16384	6553
16384	9011
16384	11468
16384	13926
16384	16383
16384	18841
16384	21298
16384	23756
16384	26213
16384	26542
16384	26542
16384	26542
16384	26542
-16384	19988
-16384	17531
-16384	15073
-16384	12616
-16384	10158
-16384	7701
-16384	5243
-16384	2786
-16384	328
-16384	-2129
-16384	-4587
-16384	-7044
-16384	-9502
-16384	-11959
-16384	-14417
-16384	-16874
-16384	-19332
-16384	-21789
-16384	-24247
-16384	-26543
-16384	-26543
-16384	-26543

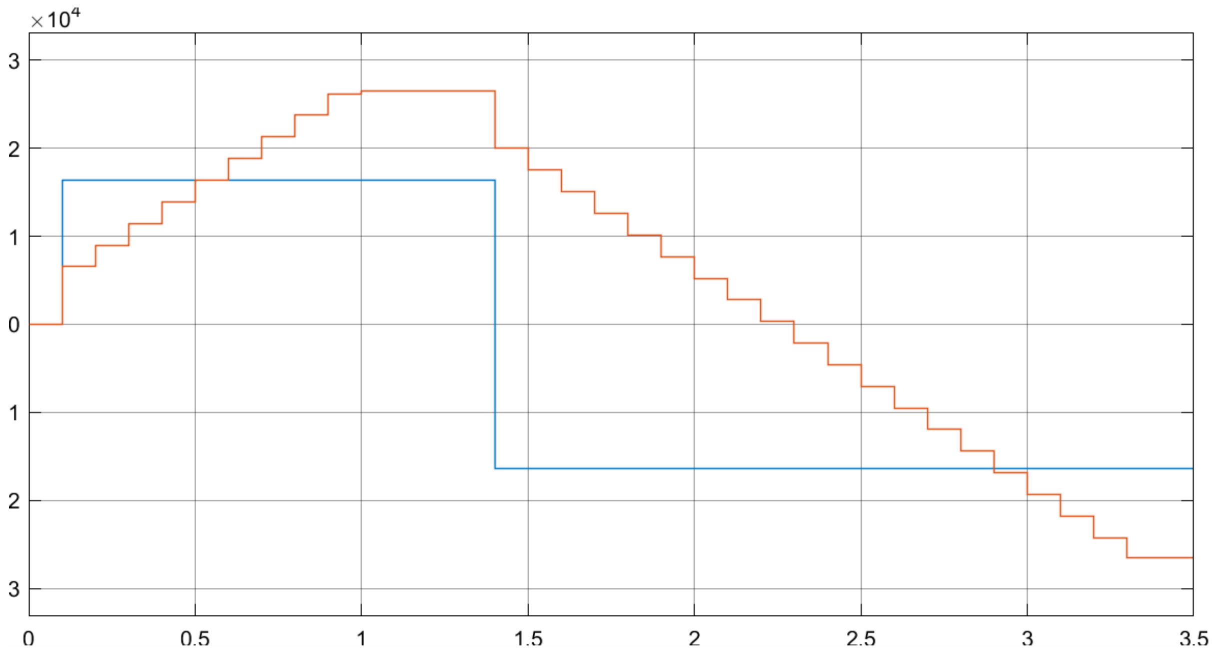


Figure 7 The comparison of input and output values of PI controller

The step input to the PI controller and output from the PI controller are as shown in the above figure. The positive peak value is seen to be 26542 from the table and negative saturation value to be -26543. Also, the positive and negative input given can also be seen from the graph that are +16384 and -16384 respectively.

6 VHDL simulation output

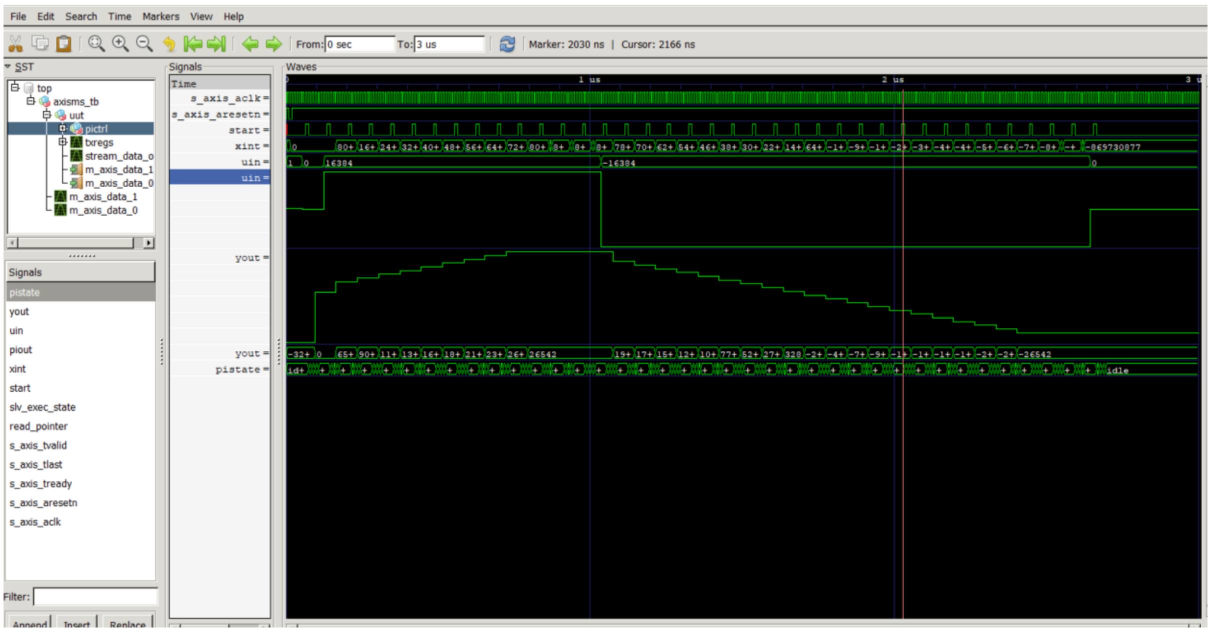


Figure 8 PI controller simulation

Complete simulation of the PI controller can be seen from the figure. The graph is similar to the graph obtained with MATLAB simulation. The input uin and output yout are made into step structure so that, it can be easily compared with the MATLAB output.

For a more detailed view, refer to the figure below.

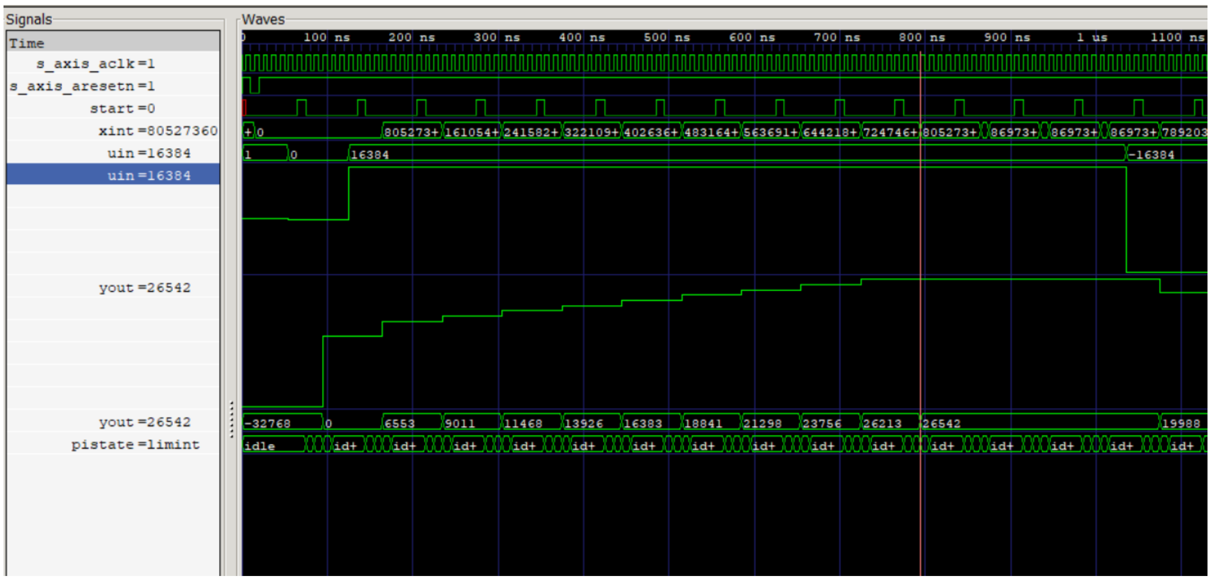


Figure 9 PI controller simulation part 1

It can be observed in the graph that the yout, output from the PI controller is saturating to a value of 26542, similar to what we have seen in the MATLAB simulation. This is when the input to the controller is a positive value of 16384. And controller is expected to hold the value unless the input is changed.

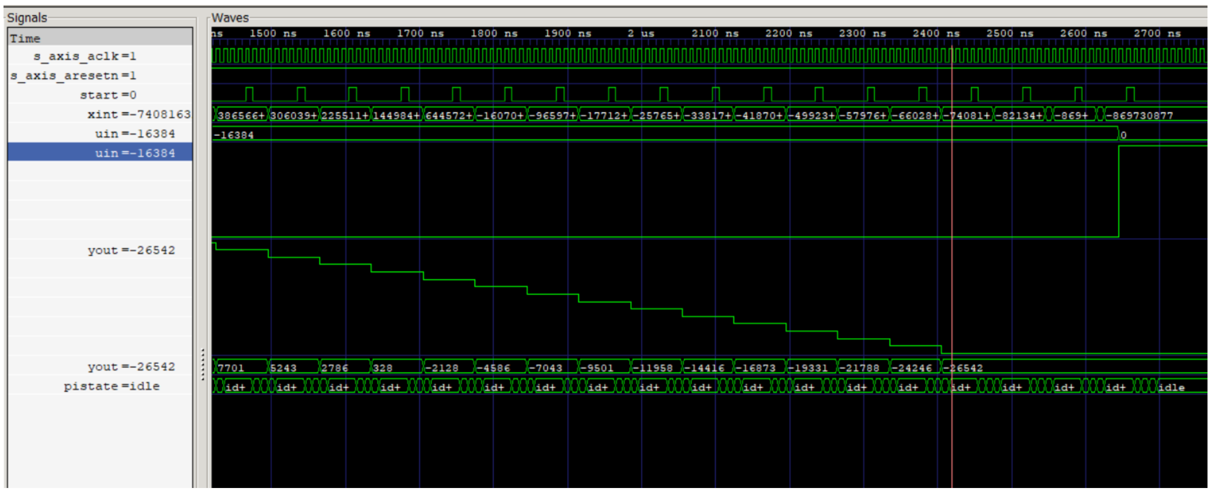


Figure 10 PI controller simulation part 2

For a negative input of -16384, it can be observed to be saturating to a value of -26542. This also is similar to the graph obtained in the MATLAB simulation. It is expected to saturate to a negative value and hold there.

7 Conclusion

PI Controller using AXIS is implemented and tested. The Value is limited to ±0.81. The output of the simulator is checked for both the positive and negative values.