Volume 2, Issue 2 (2014) 366-372 ISSN 2347 - 3258
International Journal of Advance Research and Innovation

Implementation of Kalman Filter Using Vhdl

Jolly Baliyan [*]**, Atiika Aggarwal, Ashwani Kumar**
Department of Electronics & Communication Engineering, Meerut Institute of Technology, Meerut, India

Article Info

Article history:
Received 5 March 2014
Received in revised form
20 April 2014
Accepted 28 May 2014
Available online 15 June 2014

Keywords

Kalman Filter,
FPGA,
Prediction Model,
Measurement Model, VHDL.

Abstract

The main task in object tracking is to filter the movement information from undesired dynamic objects because this information is considered as noise. To cope with these difficulties the implementation of edge segment tracking (EST) algorithm based kalman filter is presented which is used to track the desired dynamic object and to filter the noise. The estimation of current state depends on the variables i.e. time, velocity, covariance and noise mainly. Segmenting objects is capable of identifying moving objects in image sequence. One object may consist of several parts with different motion as object motion and shape are less consistent within frames. The hardware implementation of kalman filter is done on FPGA (Virtex 5) using VHDL on Xilinx ISE simulator in the range of MHz clock frequency and tested with an ADC and DAC which were integrated into the design to support analog signals at the input and output of the system.

1. Introduction

Kalman Filter follows an EST algorithm. Due to the presence of real time input there is a need of result optimization (F. A. Faruqi at el, 1980). Kalman filter is used to estimate the state of a linear system where state assumed to be distributed by a Gaussian. Kalman filter is derived from a principle which explains a property that specifies that product of two Gaussian distribution is another Gaussian distribution. Kalman Filter using state techniques as state space methods helps in simplifying the implementation of the filter in the discrete domain. As the inputs are not fixed so the location of object is shown in terms of probability. By predicting the object position from the previous information and verifying the existence of the object at the predicted position. Estimation is performed to reach to the real value by the help of sampling process which further get extended for the larger domain. Estimation Filter theory states that the state vector is estimated for a given time based upon all past measurements. It is an optimal algorithm because of its less computational requirements. There are two approaches to implement kalman filter either as hardware or software. There are two types of architecture can be possible for kalman filter and they are:

Loop Rolled Architecture

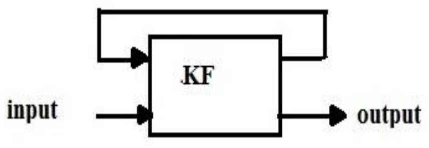

Fig: 1. Loop rolled architecture

Corresponding Author,
E-mail address: jollychaudhary1111@gmail.com

In loop architecture common hardware is using for common logic as division, multiplication, etc which is reducing the hardware (F. A. Faruqi et al, 1980).

Loop Unrolled Architecture

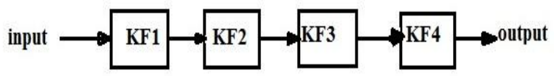

Fig: 2. Loop unrolled architecture

In loop unrolling architecture the area get increased as number of blocks is increasing due to the use of separate hardware for different states. But with that the throughput is increasing as well with the speed (F. A. Faruqi et al, 1980).

The further sections describe the implementation of EST algorithm was designed and implemented within the FPGA and tested with an ADC and DAC which were integrated into design to support analog signals at the input and output of the system.

2. Previous Work Analysis

When we analyse the previous works it is noticed that main concentration is done on the hardware area and the speed of the filter as in reference. Many hardware and software solutions have been proposed to achieve this objective. An Algebraic transformation method is proposed to reduce the differential equation and to obtain explicit expression for the filter gains which results in a substantial reduction of the computer burden involved in estimating the targets states (Y. Bar-Shalom et al, 1993). After that a mapping methodology is proposed to delivering systolic and wave front array which allow the fastest pipelining rates (S. Y. Kung et al, 1991). Many more approaches were proposed but by seeing the era the major design issues arrive of optimizing area and power consumption and reduction of mean squared error. Then Kalman Filter is introduced which reduces the mean squared error. The overall objective is to estimate x (k).The difference between the estimate of $X^{\wedge}(k)$ and x (k) itself is termed the error; $f(e(k)) = f(x(k)-X^{\wedge}(k))$

this function should be positive and increase monotonically (L. P. Maguire et al 1991). An error function which exhibits these characteristics is the squared error function and represent as f (e(k)) = $(x(k)-X^{\wedge}(k))^2$. For the ability of the filter to predict different input data over a period of time a metric is the expected value of the error function. Therefore it represent as E [f(e(k))]. This result in the mean squared error as e(t) = $E[e^2(k)]$.

3. Implementation

3.1 Functional Description of Designed Kalman Filter using (EST) algorithm

A basic top module block diagram of kalman filter is shown in figure 3. This is a looped rolled architecture of kalman filter which is used to implement the EST algorithm.

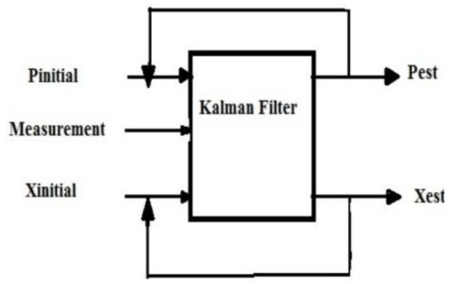

Fig: 3. A basic diagram of kalman filter

Where, Pinitial is the predicted variance
Xinitial is the predicted state
Pest is the estimated variance
Xest is the estimated state
Kalman filter has two models as process model and measurement model. The whole procedure consist three steps and they are:

- Prediction
- Measurement
- Correction

Prediction is the state which is based on the previous state.
Measurement is calculated by the help of measurement model.
Correction is estimated by the help of kalman gain, which got change with every sample.
The equation can be shown as:
x (k+1) = A x (k) + B u (k) + w (k) ,
This equation is showing the prediction state for the time (k+1) where,
A is the state transition matrix,
B is the input transition matrix,
u (k) is the uncontrolled vector which is taken zero for the simplification,
w (k) is the process additive noise
Y (k+1) = C x (k+1) + v (k+1),
This is the measurement equation where,
C is the observation matrix ,
v (k+1) is the measurement additive noise
X^(k+1) = x (k+1) + K (k+1)[Y(k+1) – x(k+1)]
This equation is showing the corrected estimated output.

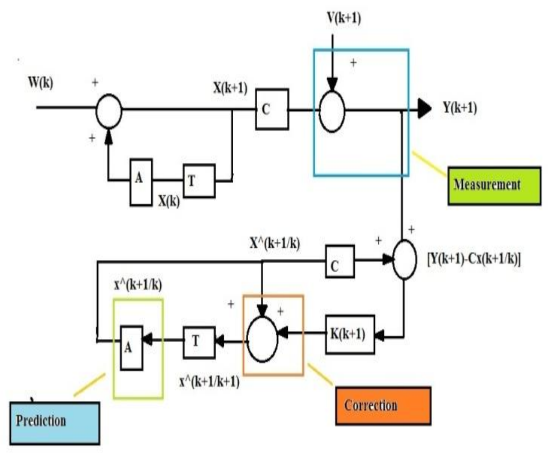

Fig: 4. Block diagram showing three basic states of kalman filter

Kalman filter equation divided into two groups:
1. Time Update
2. Measurement Update

Time update equations can be represented as:
$X^{\wedge}(k/k-1) = A_k X^{\wedge}(k-1/k-1)$
$P(k/k-1) = A_k P(k-1/k-1) A_k^T + Q(k)$
Measurement equations can be represented as:
$X^{\wedge}(k/k) = X^{\wedge}(k/k-1) + K_k [Y(k) – C_k X^{\wedge}(k/k-1)]$
$K_k = P(k/k-1)\, C_k^T (C_k P(k/k-1) C_k^T + R(k))^{-1}$
$P(k/k) = (I – K_k C_k)\, P(k/k-1)$
Where, X^(k/k-1) is predicted state
P(k/k-1) is predicted variance
X^(k/k) and X^(k-1/k-1) are updated state for (k-1) and k samples
P(k/k) and P(k-1/k-1) are updated variance for (k-1) and k samples
K_k is the kalman gain for state k
By assuming the process noise w(k) and measurement noise v(k) is uncorrelated and process noise is zero mean white noise having known covariance matrices.
E [w(k), $w(l)^T$] = Q(k) if k=l;
= zero otherwise;
E [v(k), $v(l)^T$] = R(k) if k=l;
= zero otherwise;
E [w (k), v (k)] = zero for all values of k and l
Where Q(k) is process covariance noise and R(k) is measurement covariance noise. As we the initial value of both mean and covariance matrix are unknown so we are assuming the initial value of state as
X^(0/0) = E{x(0)} and
P (0/0) = $E[\{x(0) – X^{\wedge}(0)\}\{x(0) – X^{\wedge}(0)\}^T]$
$E[||x(k+1) – X^{\wedge}(k+1)||^2] = E[\{x(k+1)-X^{\wedge}(k+1)\}^* \{x(k+1)-X^{\wedge}(k+1)\}^T]$

A. Derivation of implemented ORDP algoritm

The estimation of state X^(k+1) based on the observations up to time k, z1,z2…,zk, namely is considered (M. Munu et al, 1992).
X^(k+1) /Zk .
X^(k+1/k) = E[x (k+1)/z1,… z k] = E[x(k+1)/Zk]
X^(k+1/k) = E[x (k+1)/Z k]
= E[Ax(k)+Bu(k)+w(k)/Zk]

$= AE[x(k)/Zk] + Bu(k) + E[w(k)/Zk]$

$X\text{^}(k+1/k) = A\,X\text{^}(k/k) + Bu(k)$

$P(k+1/k) = E[\{x(k+1) – X\text{^}(k+1/k)\}\{x(k+1) – X\text{^}(k+1/k)\}^T/Zk]$

$= E[\{x(k) – A\,X\text{^}(k/k)\}\{x(k) – A^T X\text{^}(k/k)^T\}]$

$= AP(k/k)\,A^T + Q(k)$

$X\text{^}(k+1/k+1) = K'_{k+1}X\text{^}(k+1/k) + K_{k+1}Y(k+1)$

Where K'_{k+1} and K_{k+1} are weighting or gain matrices

$E[X\text{^}(k+1/k+1)] = E[K'_{k+1}X\text{^}(k+1/k) + K_{k+1}Y(k+1)]$

$= E[K'_{k+1}X\text{^}(k+1/k) + K_{k+1}C(k+1)\,x(k+1) + K_{k+1}v(k+1)]$

$= K'_{k+1}E[X\text{^}(k+1/k)] + K_{k+1}C(k+1)^* E[x(k+1)] + Kk+1\,E[v(k+1)]$

$E[X\text{^}(k+1/k)] = E[A\,X\text{^}(k/k) + Bu(k)]$

$= A\,E[X\text{^}(k/k)] + B\,u(k)$

$= E[x(k+1)]$

$E[X\text{^}(k+1)] = E[K'_{k+1} + K_{k+1}C]E[x(k+1)]$

$K'_{k+1} + K_{k+1}C = I$

Or $K'_{k+1} = I – K_{k+1}C$

$X\text{^}(k+1/k+1) = (I – K_{k+1}C)\,X\text{^}(k+1/k) + K_{k+1}Y(k+1)$

$= X\text{^}(k+1/k) + K_{k+1}[Y(k+1) – C\,X\text{^}(k+1/k)]$

$P(k+1/k+1) = E[\{x(k+1) –X\text{^}(k+1/k+1)\}\{x(k+1) – X\text{^}(k+1/k+1)\}^T/Zk]$

$= (I – K_{k+1}C)\,E[\{x(k+1) –X\text{^}(k+1/k)\}\{x(k+1) – X\text{^}(k+1/k)\}^T](I –K_{k+1}C)^T + K_{k+1}E[v(k+1)v(k+1)^T]K_{k+1}^T + 2(I – K_{k+1}C)\,E[\{x(k+1)-X\text{^}(k+1/k)\}v(k+1)^T]K_{k+1}^T$

And with

$E[v(k+1)\,v(k+1)^T] = R(k)$

$E[\{x(k+1) – X\text{^}(k+1/k)\}\{x(k+1) – X\text{^}(k+1/k)\}^T]$

$= P(k+1/k)E[\{x(k+1) – X\text{^}(k+1/k)\}\,v(k+1)^T] = 0$

We get

$P(k+1/k+1) = (I – K_{k+1}C)\,P(k+1/k)\,(I – K_{k+1}C)^T + K_{k+1}Q(k+1)\,K_{k+1}^T$

$X(k)= [X1(k), X2(k), X3(k), X4(k)]^T$

$Y(k)= [Y1(k), Y2(k)]^T$

$W(k) = [0, U1(k), 0, U2(k)]^T$

$V(k) = [V1(k), V2(k)]^T$

$Pl(k/k-1) = A\,P(k-1/k-1)\,A^T + Q(k-1)$

$X\text{^}l(k/k-1) = \mathbf{A}X\text{^}(k-1/k-1)$

$X\text{^}(k) = C\,X\text{^}l(k/k-1)$

$G(k) = Pl(k/k-1)\,C^T[CP1(k/k-1)\,C^T + R(k)]^{-1}$

$X\text{^}(k/k) = X1(k/k-1) + G(k)[Y(k) – X\text{^}(k)]$

$P(k/k) = Pl(k/k-1) – G(k)\,C\,Pl(k/k-1)$

Where

G(k) = , R(k) =

P(k/k) =

P1(k/k-1) =

Q(k) =

$X\text{^}1(k/k-1) = [X1_1\ \ X1_2\ \ X1_3\ \ X1_4]$,

$X\text{^}(k/k) = [X1\ \ X2\ \ X3\ \ X4]$,

$Y(k) = [Y1, Y2]$

Pl (k/k-1) is the priori error covariance estimate, **X**1(k/k- 1) is the priori state estimate, Y(k) is the output estimate, G (k) is the Kalman gain, X (k/k) is the posterior state estimate, and

P (k/k) is the posterior error covariance estimate. $Q(k) = E[W(k)W^T(k)]$ is the system noise covariance matrix and $R(k) = E[V(k)V^T(k)]$ is the measurement noise covariance matrix $\sigma_1^2 = E[U_1^2(k)]$ and $\sigma_2^2(k) = E[U_2^2(k)]$ are the variances of T multiplied by the radial and angular acceleration respectively and $\sigma_\rho^2(k) = E[V_1^2(k)]$ and $\sigma_{\boldsymbol{\theta}}^2(k) = E[V_2^2(k)]$ are the variances of T multiplied by the radial and angular measurement noise respectively. The tracking systems under consideration utilize sensors that provide measurements of range and bearing. Vehicle modelling is related to process model which includes two variables range and bearing. Present model is designed to track an object moving with constant speed, hence there should be four variables range, rate of change of range, bearing and rate of change of bearing. Sensor modelling is related to measurement model which includes only two variable range and bearing.

3.2 Architecture Hardware Design Approaches of Kalman Filter Implementation on FPGA

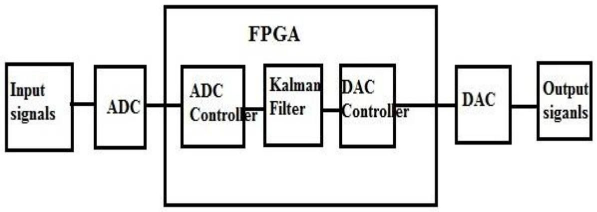

Fig: 5. Block diagram of complete system

This shows the complete kalman filter system, including an FPGA based kalman filter, analog to digital converter (ADC), and digital to analog converter (DAC). The goel of this part of the project is to produce a system where data can be streamed into and out of kalman filter as analog signals and processed into real time. The ADC connects of the kalman filter by the ADC controller, designed within the FPGA. The ADC controller directs the ADC when to take a sample of the analog input , and sends the digital value to the input of kalman filter . Like the ADC, the DAC connects to the kalman filter using a FPGA design, the DAC controller. This module sends the digital output of the kalman filter to the DAC, and then instructs it when to output that value as an analog signal, this completes the system.

The responsibility of this kalman filter design is to reject higher frequencies signals from passing through the system, but allow lower frequencies to pass unaffected.

3.1.1 kalman Filter

The kalman filter is an important part of this project. The purpose of kalman filter is to use measurements observed over time, which contains random variations of noise, and produce a value that is accurate to the true values of the measurements. It does this by predicting a value, estimating the uncertainty of the predicted value, and computing a weighted average of the predicted value and calculated value. The kalman filter first predicts the next value as well as the error covariance. When the next value comes into the filter, the kalman gain is computed, the estimate is updated with observed value, and the error covariance is updated. This helps to get rid of the noise within the signal.

A kalman filter is different from other filters such as low pass or high pass filters. These filters are linear, time invariant systems which are designed with frequency response in mind. These filters tend to be single input single output systems. In this the kalman filter we are designing could be replaced by one of these filters because it has all the characeristics just described. However in this we show how an FPGA based kalman filter could be beneficial for the locator device. The kalman filter in the locator device is designed with the characteristics of a normal kalman filter. These characteristics involve being a multiple input, multiple output system. Also they are linear, time variant systems which are designed with a mean square error approach.

The way in which Locator device uses its kalman filter is by using multiple input signals. Each of these signals by themselves could be ued to determine location. However the system cannot rely on any one source because if different scenarios, each of these input signals could be contaminated by different amounts of noise. So, kalman filter takes the information from all these signals, uses it to reduce the noise and produces its best estimate for the location of the first responders. A kalman filter is used to read the transmitter and get rid of the noise and return back the exact location of a particular person.

3. 2.2 Designing the Fpga Based Kalman Filter

Since the design was quite complicated, it was determined that the best approach was to break the design down into small pieces.

3. 2.2.1 Matrix Multiplications

The first obstacles presented were the three matrix multiplications. There were certain instances where 3x3 matrix multiplications are required. Some of the multipliers are equipped with by single multiplications. So there is not enough multipliers are available in the FPGA. We came to the conclusion that we had to multiplex what values are being multiplied at certain times, so that each multiplier could be used more than once. The entire 3x3 matrix remains the same for these calculations but the row of 3 is what is getting multiplexed for another 3 x 3 matrix. A module was created to perform the 9 multiplications between a 1x3 matrix and a 3x3 matrix. In order to perform a matrix multiplication with two 3x3 matrices, this module needs to be used 3 times. For the multiply function we have used the logic of complementing the negative value then we get the 32 bit product of two 16bit operand. This multiplier function is used for the matrix multiplication.

prod1 <= multiply **(row1, col11)** + multiply **(row2, col21)** + multiply **(row3, col31)**;

prod2 <= multiply **(row1, col12)** + multiply **(row2, col22)** + multiply **(row3, col32)**;

prod3<=multiply **(row1, col13)** + multiply **(row2, col23)** + multiply **(row3, col33)**;

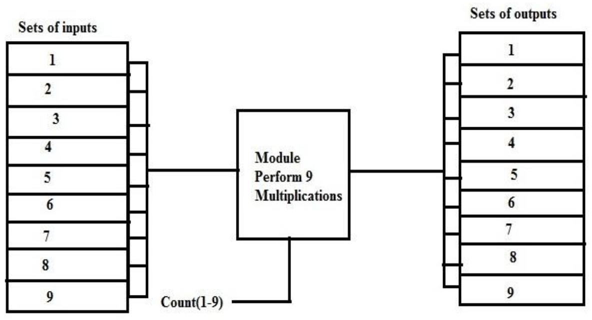

Fig: 6. Multiplexing for count 3-8

3. 2.2.2 Division

Another task is we dividing two values within the kalman filter. Performing division is adifficult task because it taks a lot VHDL code and uses a lot of resources. It was decided that the best approach was to use the built in core generator in the Xilinx software that the VHDL design was being written in. The core generator can create a number of different functions and it uses an efficient amount of resources. Once we created this module, weworked to include it within our Kalman filter design. The schematic of the function can be seen below:

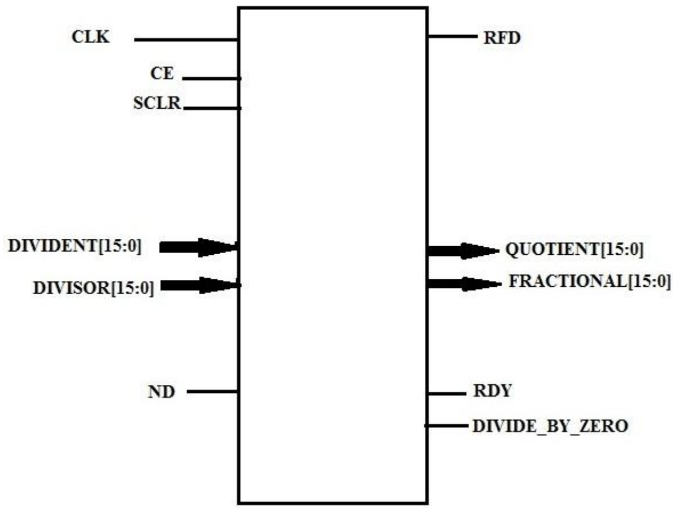

Fig: 7. Divide function

Two 16 bit signed values are entered into the function. A 16 signed bit quotient is output, as well as a remainder. This module is very useful within our design, and was quite simple to implement.

3. 2.2.3 Using Registers To Store Previous Values

The use of registers was very important to the design. Since the design uses previous values needed to be stored in registers. Also, the values in these registers have to be loaded into the design along with the input. Without using registers and loading values in on each clock value, the design would cause a continuous loop. This happens because the output changes, the current calculations would change causing the output to change again, and this would keep happening. So, once we knew we neede to use registers to store previous values, a block diagram was first to better understand how this could be done.The block diagram proved to be very helpful. A simple project was first created to make sure the process worked before it was added to the overall project. This turned out to be a success and we could clearly see values being stored in registers, and then loaded from registers. The block diagram for using the registers can be seen below. On the rising edge of the clock, or for testing purposes, when a button is pressed, flip flop load the previous output as well as the current input. After going through the next state logic, the output values are stored in the registers and remain their until the next load.

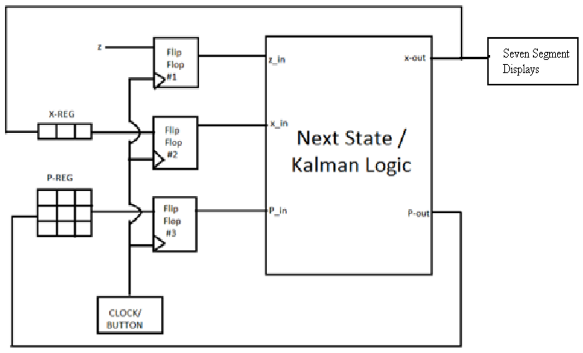

Fig: 8. Main block diagram

3.2.2.4 Using the Kalman Filter with Real Time Signals

The goal of this project is to be able to send an analog signal into the project, and output the resulting analog signal . To successfully do this, it is necessary to use an analog to digital converter (ADC) and digital to analog converter (DAC) to stream data into and out of the system using analog signals.

```
-- Control for temp
process(clk)
begin
if rising_edge(clk) then
if count = 0 then
if load = '1' then
temp <= data;
elsif currentstate = Send then
temp <= temp(14 downto 0) & '0';
end if;
end if;
end if;
end process;
-- Next State Logic
process(currentstate, load, regcount)
begin
case currentstate is
when Idle =>
if load = '1' then
nextstate <= Low;
else
nextstate <= Idle;
end if;
when Low =>
nextstate <= Send;
when Send =>
if regcount = 15 then
nextstate <= High;
else
nextstate <= Send;
end if;
when High =>
nextstate <= Idle;
end case;
end process;
```

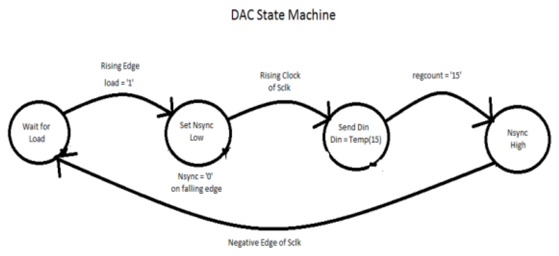

Fig: 9. State machine of DAC

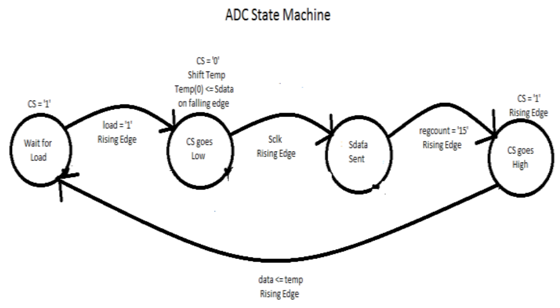

Fig: 10. State machine of ADC

```
--Next state logic
process(currentstate, load, regcount)
begin
case currentstate is
when Idle =>
if load = '1' then
nextstate <= CSLow;
else
nextstate <= Idle;
end if;
when CSLow =>
```

```
nextstate <= Receive;
when Receive =>
if regcount = 15 then
nextstate <= CSHigh;
else
nextstate <= Receive;
end if;
when CSHigh =>
nextstate <= Idle;
end case;
end process;
-- Control for temp
process(clk)
begin
if rising_edge(clk) then
if nextstate = Receive then
if count = 0 then
temp <= temp(14 downto 0) & Sdata;
elsif count = 3 then
temp(0) <= Sdata;
end if;
end if;
end if;
end process;
-- Load 12 data bits to data
process(clk)
begin
if rising_edge(clk) then
if currentstate = CSHigh then
data <= temp(11 downto 0);
end if;
end if;
end process;
```

CS is what controls when a sample is taken. CS is active low and tells the ADC to create a 16 bit value out of the analog sample. Like the DAC, we created a load signal to tell the ADC controller that an input is desired. When load is high, CS goes low on the rising edge of sclk. This allows for the setup time to be achieved before the first value is input on the falling edge of the clock. Temp shifts in one bit at a time on the rising edge of sclk and after 15 cycles, temp is ready to output a 16 bit value, and CS is sent high. The values are available on the falling edge of sclk, but taking them on the rising edge assures that they are valid. The only bit that is taken on the falling edge is the first bit, and this is because it is sent along with the second bit on the first falling edge. The values of Sdata were shifted into temp.

The first step to this process is using a DAC to output the values of the Kalman Filter as an analog signal. The first thing we had to do was to choose which DAC to work with. The DAC is of 12-bit instead of a 16-bit DAC means that there is a loss of precision, but since only the least significant bits are lost, the difference in precision does not affect the performance of the kalman filter.

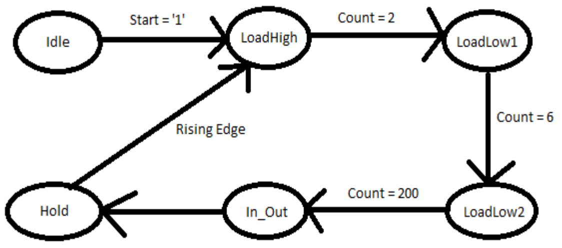

Fig: 10. State machine of ADC to DAC

```
-- Next State Logic
process(currentstate, start, count25K)
begin
case currentstate is
when Idle =>
if start = '1' then
nextstate <= In_Out;
else
nextstate <= Idle;
end if;
when In_Out =>
if count25K = 20 then
nextstate <= LoadHigh;
else
nextstate <= In_Out;
end if;
when LoadHigh =>
if count25K = 22 then
nextstate <= LoadLow;
else
nextstate <= LoadHigh;
end if;
when LoadLow =>
if count25K = 26 then

nextstate <= Hold;
else
nextstate <= LoadLow;
end if;
when others =>
if count25K = 2000 then
nextstate <= In_Out;
else
nextstate <= Hold;
end if;
end case;
end process;
```

4. Conclusion

In conclusion, we have created a successful Kalman Filter, which interfaces with an ADC and DAC to form a complete system that streams analog data in and out. We also created an ADC controller and DAC controller so that the Kalman Filter, ADC and DAC could be integrated together and used for testing purposes.A complete system like the one we have built can be altered, and added onto, to perform the tasks of the Kalman Filter in the PPL(Precision Personal Locator) system, and can be included within the implementation of the actual system to process the data in real time. What all of this means is that another group can learn from everything that has been documented here to enhance our design to support a more complicated version of a Kalman Filter.

References

[1] F.A. Faruqi and R.C. Davis, Kalman Filter design for target tracking, IEEE Trans. Aerosp. Electron. Syst., AES-16, 500-508, 1980

[2] Y. Bar-Shalom and X. R. Lin, Estimation and tracking: principles, techniques, and software, Artech House, 1993, 417-483

[3] S. Y. Kung and J. N. Hwang, Systolic array designs for Kalman Filtering, IEE Trans., Signal Processing, 171-182, 1991

[4] L. P. Maguire and G. W. Irwin, Transputer implementation of Kalman Filter, IEEE Proc., 138, 355-362, 1991.

[5] M. Munu, I. Harrison, D. Wilkin, and M. S. Woolfson, Comparison of adaptive target-tracking algorithms for phased-array radar, IEE Proc. F. Commun, Radar and Signal, 139, 336-342.

[6] D. P. Atherton and H. J. Lin, Parallel implementation of IMM tracking algorithm using transputers, IEE Proc.-Radar, Sonar Navig., 141, 325-332, 1994

[7] J. M. Jover, T. Kailath, A parallel architecture for Kalman Filter measurement update and parameter estimation, Automatica, 22, 32-57, 1986

[8] Song Ci. Sharif, H (2005) Performance Comparison of Kalman Filter based approaches for energy efficiency in wireless sensor networks, IEEE conf.: on Computer Systems and Applications.