BEAMFORMING ULTRA-WIDEBAND TRANSMITTER

SAISUNDAR SANKARANARAYANAN

(B.E., ANNA UNIVERSITY)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL & COMPUTER

ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2009

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my research supervisor, Dr. Heng Chun Huat, for the guidance and support that he gave me throughout the course of this research. I thank him for providing me an opportunity to work on this project. The ideas and feedback that he shared with me are invaluable to my research. Without his encouragement, support and guidance, this work would not have been possible.

I am grateful to all administrative and technical staff for the help and support they provided me. I would like to thank from the bottom of my heart all those who have supported me through their friendship and thoughtfulness.

I would finally like to thank my parents for their unwavering support and love which has been the source of strength for me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

UWB	- Ultra-Wideband
IR-UWB	- Impulse Radio UWB
EIRP	- Effective Isotropically Radiated Power
DLL	- Delay Locked Loop
FSM	- Finite State Machine
PVT	- Process, Voltage and Temperature
VCDL	- Voltage Controlled Delay Line
SAR	- Successive Approximation Register controlled
DPA	- Digital Power Amplifier
LO	- Local Oscillator
MUX	- Multiplexer
F_{REF}	- Reference Frequency
ΔT	- Pulse Width
T_{REF}	- Reference Period
$\mathrm{V_{CTRL}}$	- Loop Filter control voltage
K_{VCDL}	- Gain of Voltage controlled delay line
I_{CP}	- Charge pump current
f_{BW}	- Loop Bandwidth
$\Sigma\Delta$	- Sigma-Delta
Vp-p	- Peak to peak voltage

ABSTRACT

This thesis presents a beamforming ultra-wideband transmitter capable of providing very fine angular resolution for imaging applications in the frequency range of 3-5GHz. This beamforming UWB transmitter employs baseband processing to save area and power. The transmitter uses a novel digital delay calibration technique to obtain the precise delay difference. The digital delay calibration system consists of a serial to parallel programming interface (SPI), a digital calibration Finite State Machine (FSM), a fractional-N sigma-delta delay locked loop (ΣΔDLL) and a (Successive Approximation Register) SAR delay locked loop. The sigma-delta delay locked loop will provide fine delay resolution as well as large delay range required in beamforming applications. This analog delay is then converted to a digital code word with the help of a SAR DLL. The FSM stores the digital code words corresponding to different delay values. These digital code words are then applied to a beam former delay subsystem to generate baseband signals with precise delay difference. The UWB pulse is generated from these signals with the help of all-digital UWB transmitters. The whole transmitter architecture is implemented in 0.13µm CMOS process. It provides a delay resolution of less than 10ps and consumes about 35.4mW of power from a 1.2V supply. The all-digital UWB transmitter implemented in the design provides the flexibility to shape the output UWB pulse and consumes 80pJ/pulse.

CHAPTER 1 : INTRODUCTION

Ultra-Wideband (UWB) technology is an emerging field of research. This field has gained a lot of focus in recent years, especially after the Federal Communications Commission (FCC) approved the use of 3.1-10.6GHz band for unlicensed communication using UWB. UWB has many appealing features such as robustness, flexibility, high data-rate capability as well as high-precision ranging capability. These features make it very attractive for applications like radar, imaging, wireless USB, sensor networks, RFID tags etc. In radar and imaging applications, Impulse Radio UWB (IR-UWB) is carving a place for itself because of the ranging resolution it can provide. UWB is emerging as a low cost and high data rate alternative in the field of wireless communications [1]. This chapter provides an introduction to UWB, its applications and outlines the important contributions of this thesis.

1.1 UWB and its applications

Recognizing the potential advantages of ultra-wideband (UWB) communication, Federal Communications Commission (FCC) issued a report that permitted the use of 3.1 to 10.6GHz spectrum for UWB wireless communication [9]. According to the FCC, a signal is considered ultra-wideband if it has a fractional bandwidth greater than 0.2 or has an absolute -10dB bandwidth greater than or equal to 500MHz.The fractional bandwidth is defined as the ratio of -10dB bandwidth of the signal to its centre frequency.

The 802.15.3a task group had categorized UWB into two categories, i.e. the Orthogonal Frequency Division Multiplexing (OFDM) standard and the Impulse Radio UWB (IR-UWB) standard. The OFDM standard has been adopted by Wi-media alliance for implementing high data rate communication. The IR-UWB standard is very useful for energy efficient designs. Low data rate transmitters with low energy consumption have been built using the IR-UWB proposal [2, 3].

Some of the applications that can benefit from low power, low cost and high data rate provided by UWB are Wireless Personal Area Network (WPAN) applications which provide wireless connectivity to all personal computing devices, consumer electronics and mobile devices. The maximum data rate that can be achieved on a RF link is given by the Shannon capacity theorem as

$$C = BW\log(1+SNR), \tag{1.1}$$

where, C is the channel capacity, BW is the bandwidth and SNR is the signal to noise ratio. The wide bandwidth offered by UWB helps to attain a high data rate which is required for high-speed data transfer applications. The low power and low data rate applications like sensors and biomedical applications exploit the low energy consuming architectures of the IR-UWB. It is very useful in RFID tags for patient and asset tracking in hospitals. In radar and imaging applications, IR-UWB is very attractive because of the ranging resolution it can provide. It is very useful for range finders and biomedical imaging [5]. Ranging resolution is the minimum separation distance between the two objects that can be detected by the radar system. The ranging resolution is given by

$$R = \frac{c}{2BW} \tag{1.2}$$

where, *BW* is the bandwidth of the signal and *c* is the speed of light.

Since UWB signals provide the user a potentially wide bandwidth, the ranging resolution that can be obtained could be very good. By utilizing the 7.5 GHz bandwidth from 3.1-10.6GHz, resolution as low as 2cm can be achieved.

UWB is considered for a wide range of applications as shown in Figure 1.1 [33].

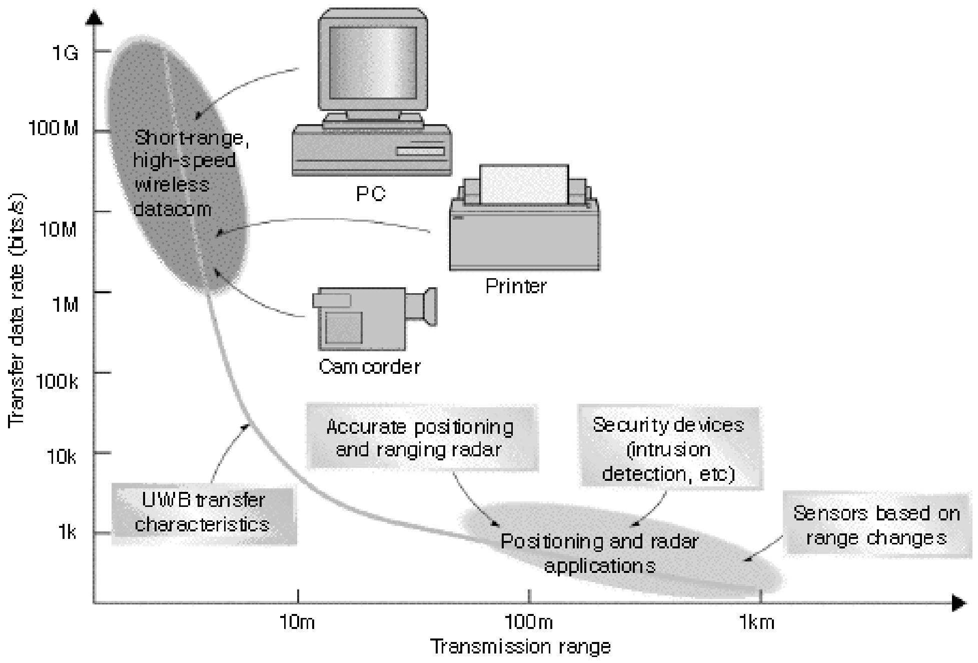


Figure 1.1: Applications of UWB in short range wireless communications and ranging [33]

UWB and radar based imaging are used in military applications [4]. UWB imaging is also used for detecting cardiac and pulmonary signals generated from the heart and lungs [8].

1.2 Motivation

UWB has a very high-precision ranging capability. This is predominantly the driving force for using UWB in imaging and ranging applications. In a typical imaging system, a transmitter would transmit an electromagnetic pulse train towards objects of interest. A receiver would collect the signals that are reflected and/or scattered from the objects as shown in Figure 1.2.

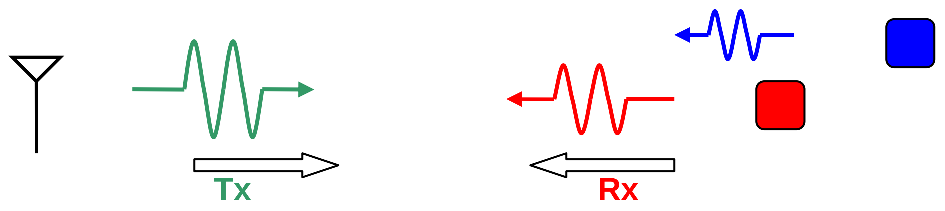


Figure 1.2: Typical imaging system

The magnitude and shape of the received signals and their time of arrival are used to identify the target and its relative distance and speed respectively [5]. Beamforming helps to focus the beam in a particular direction as indicated in Figure 1.3. This helps in improving the spatial resolution of the image. The electromagnetic beam can be scanned mechanically or electronically to improve spatial coverage.

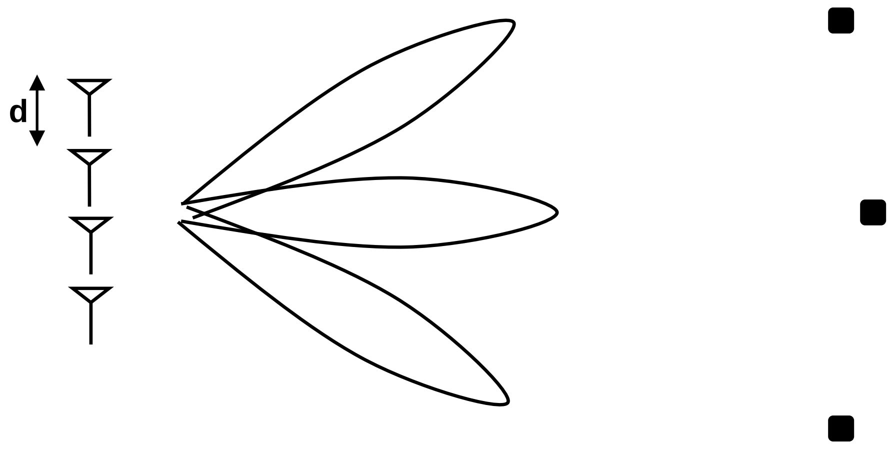


Figure 1.3: Beamforming in imaging system

An antenna array capable of electronic beam steering can adapt and focus its Radio Frequency (RF) beam at specific directions [1]. This helps to locate targets in radar and imaging systems. Angular resolution is defined as minimum angle by which the beam can be steered. Systems capable of providing very fine beam steering are needed for the high precision imaging applications. This thesis presents a beamforming ultra-wide band transmitter architecture capable of providing high angular and depth resolutions required for the imaging systems in the frequency range of 3-5GHz.

1.3 Thesis contribution

The major contributions of this thesis are

1. A novel beamforming ultra-wideband transmitter architecture capable of achieving very fine beam steering angles.
2. A novel digital delay calibration technique that uses a digital FSM, sigma-delta delay locked loop, SAR delay locked loop and a vernier delay line to generate the precise delay required for beam forming. This helps to calibrate and control the beam angle and the centre frequency of the UWB transmitter.
3. An improved pulse shaping logic circuit which provides additional flexibility to control the shape of the output UWB signal.

1.4 Thesis organization

This thesis presents a beamforming UWB transmitter implemented in 0.13µm IBM CMOS process. The organization of this thesis would be as follows. Chapter 2 provides the basic theory and advantages of beam forming in UWB. Chapter 3 reviews the existing beamforming UWB transmitter architectures. The proposed beamforming UWB transmitter architecture is presented in chapter 4. Chapter 5 presents the CMOS implementation of the transmitter and chapter 6 concludes with a summary.

CHAPTER 2 : UWB BEAMFORMING

2.1 Advantages of UWB Beamforming

The power that can be emitted in the approved UWB band from 3.1GHz to 10.6GHz is very limited. The FCC limits for indoor communications are indicated in Table 2.1 [9] while the indoor emission mask of UWB is shown in Figure 2.1.

Table 2.1: FCC emission limits for indoor UWB [9]

Frequency in MHz	EIRP in dBm
960-1610	-75.3
1610-1990	-53.3
1990-3100	-51.3
3100-10600	-41.3
Above 10600	-51.3

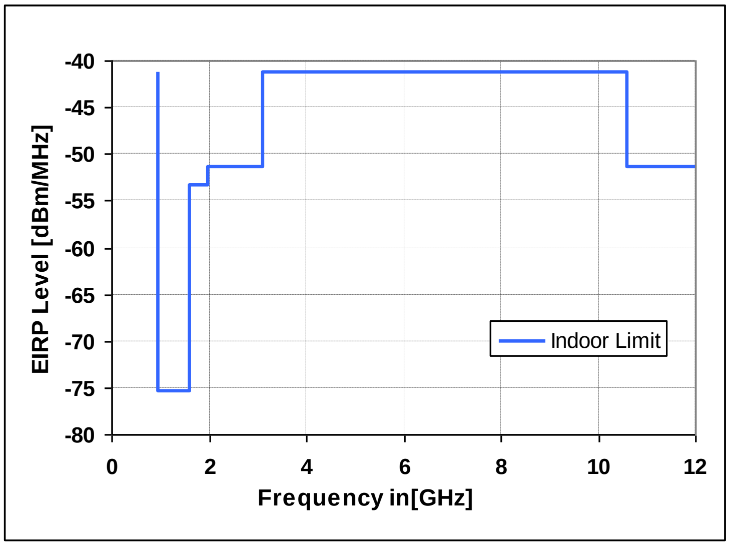


Figure 2.1: Indoor UWB emission mask

In order to avoid interference to other existing narrow band systems, FCC has restricted the output power that can be transmitted from an UWB system. Because of this low output power restriction, the coverage in an UWB system is limited to only a few meters. However, this disadvantage of UWB can be overcome by using beamforming.

For a given power level at the receiver, the power that has to be generated in a phased array system is lower than in an isotropic system. In an N-element transmitter radiating P watts of power, the total power in the focused direction will be N^2P. The N^2 factor is due to the coherent addition of the electromagnetic waves in the desired direction [10]. Beamforming can provide 20log(N) improvement in transmitter EIRP where N is the number of antenna elements used in the system.

Besides improving the range, beamforming can also minimize the interference to other narrow band systems. It can also avoid narrow band interferers at the receiver [10]. Beamforming helps to obtain the theoretically achievable highest data rate. The range and data rate obtained in beamforming transmitters are higher than their single antenna counterparts [11].

When compared to narrow band systems, beamforming in UWB also provides an additional degree of freedom in choosing the antenna spacing. In case of a narrow band system the antenna array factor is given by the equation

$$AF(\theta) = \frac{\sin(N(kd\sin\theta - \beta)/2)}{N\sin((kd\sin\theta - \beta)/2)} \qquad (2.1)$$

where, θ is the polar co-ordinate, N is the number of antenna elements, d is the spacing between the antenna elements, β is the angle at which the main lobe of the beam is focused and k is the propagation vector of the transverse electromagnetic wave and it is related to the wavelength(λ) by

$$k = \frac{2\pi}{\lambda} \tag{2.2}$$

This is a very well known equation for the antenna pattern of a narrow band system. An analysis of this equation would show that the antenna pattern would have side lobes and grating lobes. To obtain minimum side lobes, the antenna spacing has to be half-wavelength ($\lambda/2$).

The array pattern equation for a Gaussian UWB signal is obtained in [13-15] and the approximate array pattern expression for the UWB signal is indicated in [4] and the equation is as follows

$$AF(\theta) = \frac{erf\,(\sqrt{\pi}L\sin\theta/(2\Delta Tc))}{(\sqrt{\pi}L\sin\theta/(2\Delta Tc))} \tag{2.3}$$

where, θ is the polar co-ordinate, c is the velocity of light, ΔT is the pulse width and L is the antenna array length. For an antenna array with N elements each separated by a distance of d, the array length is given by

$$L = (N-1)d \tag{2.4}$$

The polar plot of the array factors given by equations 2.1 and 2.3 for the narrow band and the broad band systems respectively were obtained using MATLAB and the result is as shown Figure 2.2.

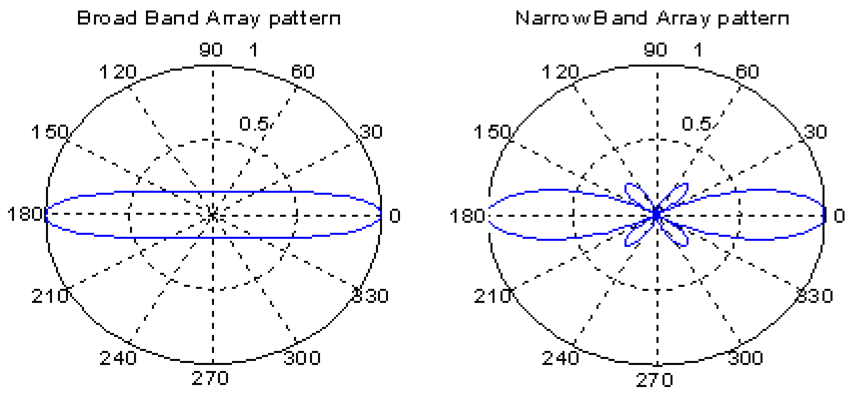


Figure 2.2: Comparison of UWB and narrow band array pattern

It can be clearly seen that the UWB beamforming array pattern does not have the grating lobes and side lobes that are present in narrowband systems [4]. Therefore, it is no longer necessary to restrict the antenna spacing to half-wavelength spacing. Larger spacing can be used for obtaining higher spatial selectivity [5].

2.2 UWB beamforming system basics and requirements

The properties of UWB beamforming systems have been discussed in the literature [13-15]. In an UWB beamforming system consisting of a linear array of antennas, the scanning angle depends on the spacing between antenna elements and the radio pulse width (ΔT) of the UWB signal. The angle between the direction where the beam is focused normal to the antenna array is indicated as θ as shown in Figure 2.3.

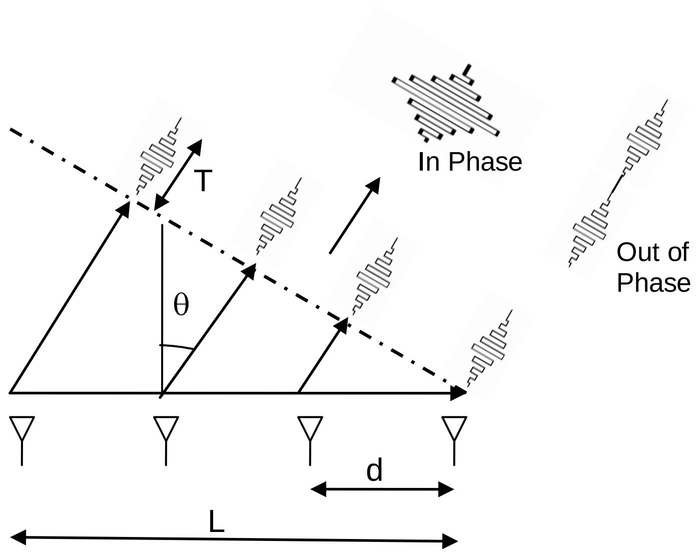


Figure 2.3: UWB beamforming system.

In a linear array of antennas, with a spacing of *d* between the antenna elements, in order to achieve a maximum signal in the direction *θ*, the relative delay between the signals fed to the adjacent antenna elements is given by

$$\Delta = (d \sin\theta)/c \qquad (2.5)$$

where, *c* is the velocity of electromagnetic wave in free space.

UWB arrays require true time-delay elements to steer the RF beam as indicated in the Figure 2.4. The delay difference between the adjacent antenna paths decides the scanning angle.

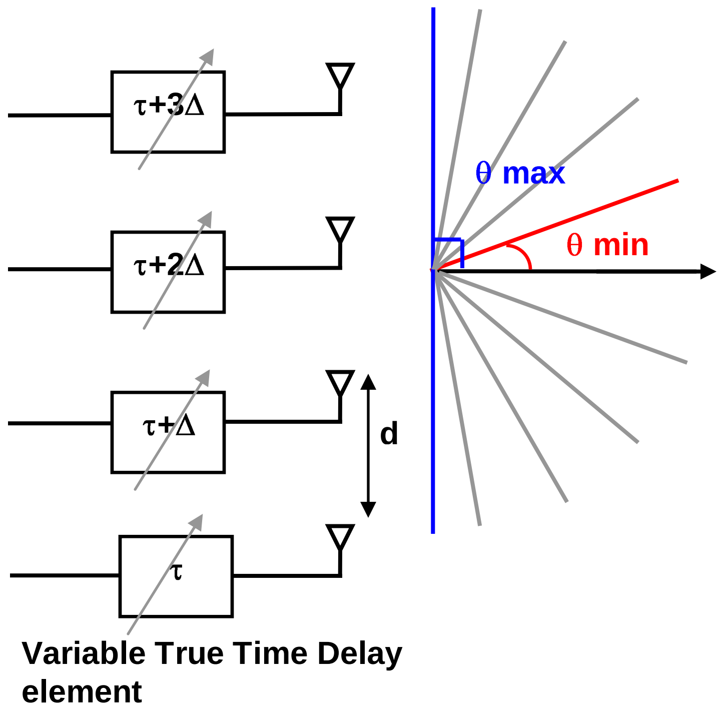


Figure 2.4: UWB array with true time-delay element

The requirements for the design of true time-delay elements for an UWB beam forming system were summarized as follows [1]:

1) The relative time-delay between adjacent antenna elements should be a fraction of the pulse width of the UWB radio pulses to steer a fine scanned angle. Longer delays are required for wider scanned angles.

2) The true time-delay element should be variable to change the scanned angle adaptively; hence, electronic control is desirable.

3) Timing jitter or deviation of the true time-delay element should be minimized.

These requirements listed for the design of true time-delay elements, serve as guideline in selecting the architecture for beamforming UWB transmitter.

CHAPTER 3 : EXISTING BEAMFORMING UWB TRANSMITTER ARCHITECTURES

3.1 Literature Review

Electronic beam steering in UWB phased array transmitters can be done in baseband or RF. The following summarizes some of the existing beamforming UWB transmitter architectures and investigates their advantages and disadvantages.

In design [16], beamforming was implemented by connecting the true time-delay elements between the transceiver and power splitter/combiner. The delay elements were implemented using passive elements like RC or transmission line components. But obtaining a large delay using transmission line components consumes a large area and is not economical.

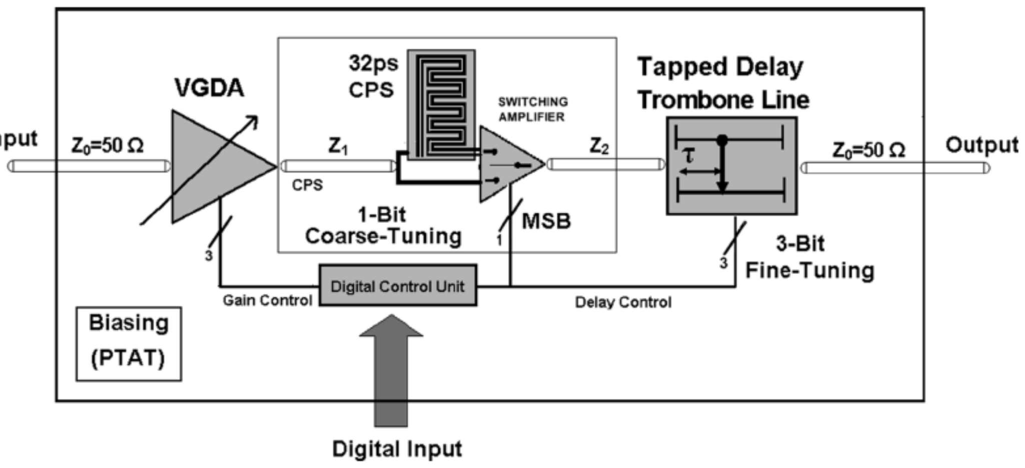


Figure 3.1: UWB beamforming system from [4]

The UWB beam former in [4], implemented in 0.18μm BiCMOS SiGe process can obtain a ranging resolution of 20mm and an angular resolution of 7 degrees. This architecture employs trombone delay elements at the RF path as depicted in Figure 3.1. The beamformer can provide a maximum delay of 64ps with a step size of 4ps. The trombone delay elements were constructed using lumped LC. This chip has an area of 2.25mm^2 and consumes 87.5mW from a 2.5V supply. The major drawbacks of this scheme are the huge area due to inductors and the large power required for compensating the loss incurred by the LC delay elements.

Ultra-wideband timed array receiver implemented in [5] uses a path-sharing delay architecture. This beamforming system has a resolution of 15ps and can provide 11 angles with 9 degrees of spatial resolution. The delay line was implemented using fully differential constant –k LC ladder. Each ladder section provides a resolution of 7.5ps. This chip suffers from the same drawbacks as [4]. It has a die area of 9.92mm^2 and consumes 555mW of power.

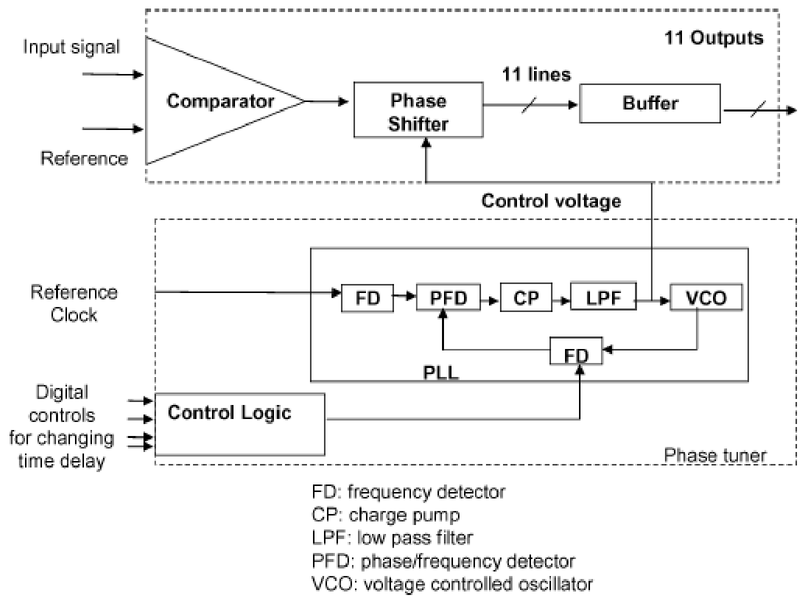


Figure 3.2: Electronic beam steering subsystem from [1]

In an effort to reduce the die area and power consumption, electronic beam steering in UWB phased array transmitter was implemented at baseband [1]. As shown in Figure 3.2, the beam steering subsystem consisted of a digital PLL that controlled a variable delay to provide the required phase shift. The input baseband signal was delayed using this beam steering subsystem. The relative delay between the paths decides the beam angle and these delayed signals were directed to an external UWB pulse forming network to produce the UWB pulses. This beam steering system was implemented in 0.25μm process and it can produce a delay in the range of 100ps to 500ps. The system can obtain a delay step size of 100ps and a minimum scan angle of 9 degrees per step with an antenna spacing of 18cm. It consumes about 100mW of power and has an area of 0.9mm^2. The main drawbacks of this architecture are the higher jitter and poorer resolution due to the employed PLL for delay control.

The UWB timed array transmitter implemented in [6] uses the path-sharing architecture at baseband to provide the delay. This architecture uses long and short inverter chain paths to obtain the delay difference between the adjacent channels. The delay resolution obtained is 180ps with a maximum delay of 880ps. With an antenna spacing of 30cm, this design can steer the beam with an angular resolution of 10 degrees within a range of ±60degrees. However, there is no mechanism to change or tune the delay of the inverter chains to compensate for PVT variations and the achievable delay resolution is limited to 180ps. In addition, it requires phase shifters to align the VCO phases which might not be trivial. The UWB pulse forming switch employed in this architecture also leads to undesirable LO leakage to the output of the transmitter.

By investigating these architectures, it is clear that beamforming UWB transmitters utilizing baseband processing is advantageous in terms of both area and power. However, they mainly suffer from poor delay resolution. On the other hand, beamforming transmitters utilizing RF processing will result in better delay resolution but they demand larger area and power consumption.

This calls for the development of a beamforming UWB transmitter with baseband processing, which can provide good delay resolution. The better the delay resolution the better will be the angular resolution. For example, with an antenna spacing of 18cm and a delay resolution of 20ps, scanning angle resolution of the order of 2 degrees can be achieved.

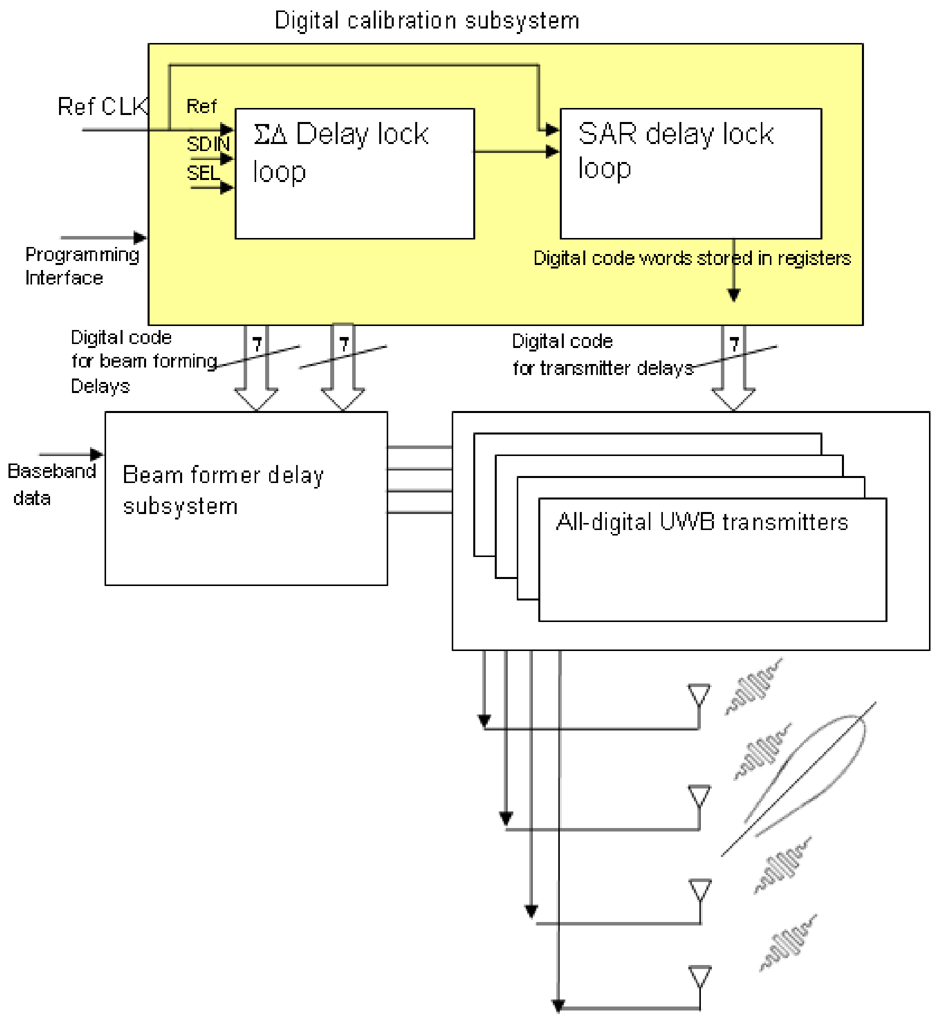
This work presents a beamforming UWB transmitter architecture dedicated to achieve good angular resolution without too much power and area penalty.

CHAPTER 4 : BEAMFORMING UWB TRANSMITTER ARCHITECTURE

This chapter presents the proposed beamforming UWB transmitter architecture. The whole transmitter was modeled and proven in the MATLAB simulation.

4.1 Beamforming UWB transmitter architecture

The proposed architecture employs baseband processing and improves the delay resolution to obtain good angular resolution. This architecture will result in a compact design. The proposed architecture is shown in Figure 4.1.



Digital calibration subsystem
Ref CLK
Ref
SDIN
SEL
ΣΔ Delay lock loop
SAR delay lock loop
Programming Interface
Digital code words stored in registers
Digital code for beam forming Delays
7
7
Digital code for transmitter delays
7
Baseband data
Beam former delay subsystem
All-digital UWB transmitters

Figure 4.1: Proposed beamforming UWB transmitter architecture.

The proposed architecture consists of a beam former delay subsystem to generate the delay required for beamforming. To generate the beamforming delays, the vernier delay line architecture was chosen. This architecture is quite popular [23, 24] to generate delay in the range of picoseconds. The concept is based on the fact that the delay difference between two paths can generate picoseconds delay. The digitally controlled delay elements used in the beam former subsystem have been designed to generate delay difference in the range of 0-600ps. These digitally controlled delay cells have been designed to give a resolution of the order of 5-10 picoseconds per bit change in the input digital code word. In order to generate the accurate delay, calibration is required.

The digital calibration circuitry consists of a fractional-N sigma-delta DLL and a Successive Approximation Register controlled DLL (SAR DLL). The sigma-delta DLL architecture can provide fine delay resolution as well as large delay range required for the beamforming subsystem. The PVT variations will be compensated through the digital calibration employing sigma-delta DLL. In order to minimize the power consumption, sigma-delta DLL is only used for the calibration phase and will be turned off subsequently. The calibration result will be captured by the additional SAR DLL. The SAR DLL will convert the analog delay calibrated by sigma-delta DLL into a digital code word through a digital calibration FSM. The code words obtained from the calibration will then be used to control the beam former delay subsystem and generate the required delay difference.

A baseband signal fed to the beam former delay subsystem will produce four delayed signals and the relative delay between the signals will decide the angle of the main beam.

These delayed signals are fed to the all-digital UWB transmitters that generate the required IR-UWB pulses. IR-UWB pulses are generated by combining the edges that are output from a transmitter delay line [2]. The centre frequency of the UWB pulse depends on the delay generated by the transmitter delay line. A delay line that is controlled by sigma-delta DLL would provide the precise delay required by these systems and the edges can be combined as in [2] to produce the IR-UWB pulses with desired centre frequency. The generated pulses need further shaping in order to be FCC compliant. Hence, the digital power amplifier with capacitively controlled pulse shaping driver architecture [3] is adopted with a new pulse shaping logic circuit to obtain the required pulse shaping. The architecture has the flexibility to control most of the parameters of the output pulse, there by giving the flexibility to obtain a FCC compliant UWB pulse. The choice of these architectures was based on considerations of power and area.

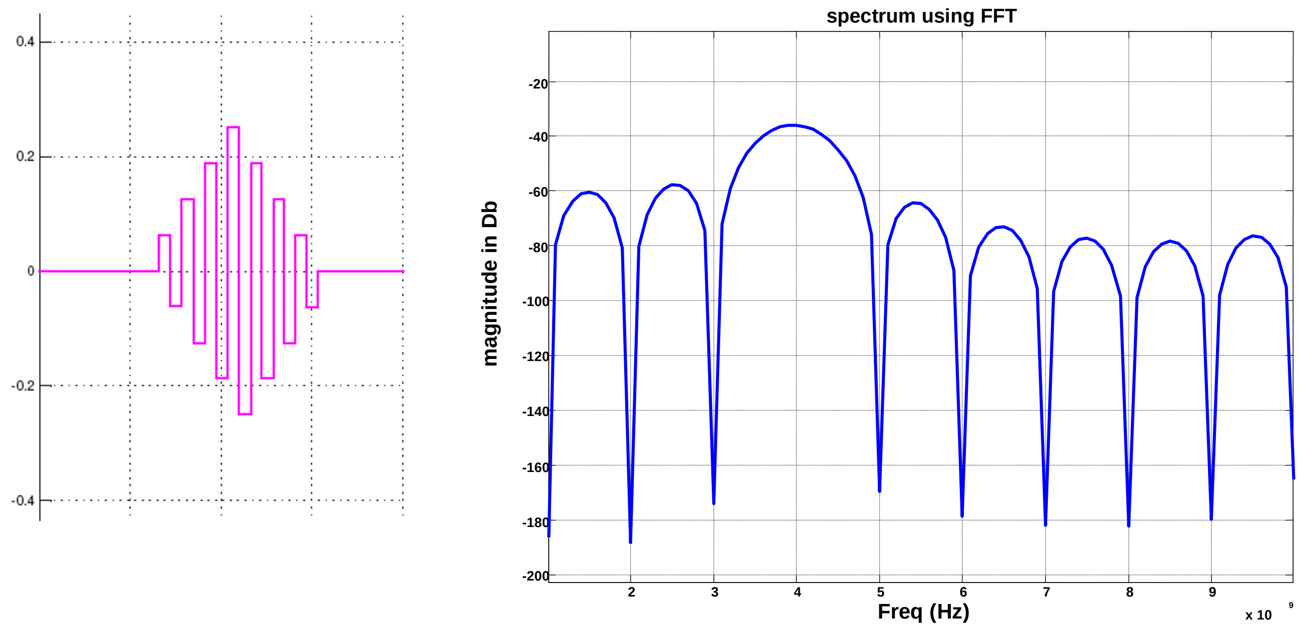


Figure 4.2: UWB pulse and its spectrum from MATLAB simulations

The whole transmitter was modeled in MATLAB and the pulse shape of the signal and its spectrum are as shown in Figure 4.2. The pulse shaping implemented in the MATLAB model was just a four level pulse shaping. The spectrum obtained is not FCC compliant. Changing the pulse shape further can result in a FCC compliant spectrum.

The MATLAB simulations of the sigma-delta DLL shows accurate control of the delay. Hence, by using this new architecture, a delay resolution of the order of 5-10ps can be obtained in the MATLAB simulation. This results in minimum steering angle of close to 1 degree with an antenna spacing of 18cm. The beam steering obtained in the MATLAB model for some angles are shown in Figure 4.3.

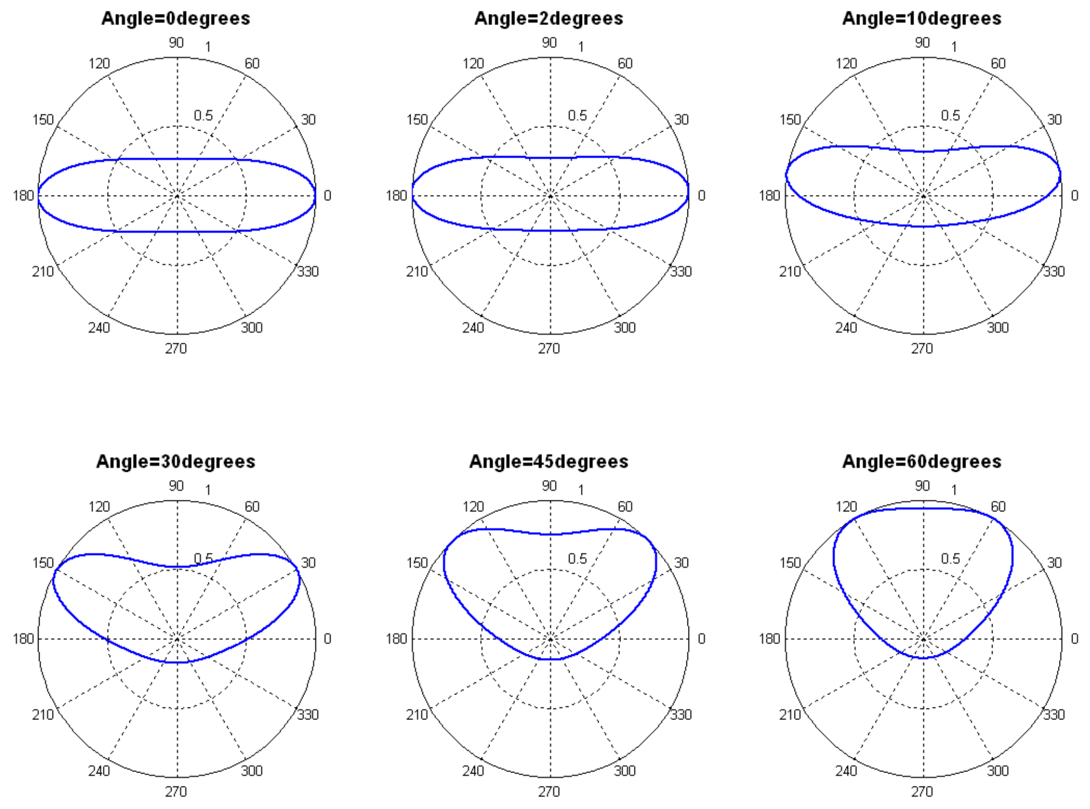


Figure 4.3: UWB beam steering from MATLAB simulations

If the CMOS delay resolution is relaxed to 20ps, the corresponding angular resolution would be 2 degrees. In the next section a detailed description of the various blocks, their workings and the MATLAB results will be presented.

4.2 Detailed description of the blocks in the proposed architecture

4.2.1 Beamforming delay subsystem

In order to steer the beam through an angle of 0 to 90 degrees, (though 90 degrees is not feasible due to the beam spreading) with a particular antenna spacing, the required delay range can be calculated using the equation (2.5). With an antenna spacing of 18cm, the required delay range is from 0-600ps.To get an angle of 0, the relative delay should be zero ps. This is not achievable using a single delay cell. Using a single delay element to generate the beamforming delay limits the delay resolution to the minimum delay achievable in the process. Hence, the vernier delay line architecture is adopted to overcome this limitation. In the vernier delay line architecture, the delay resolution depends on the delay difference between the paths. It depends on how fine the delay in a path can be tuned with respect to the other. The delay difference between the delay cells determines the delay resolution. Since a very fine tuning of the delay is needed, a digitally controlled current starved inverter was used. Current starved inverter can generate the required fine delay. The delay tree structure used in this design is shown in Figure 4.4.

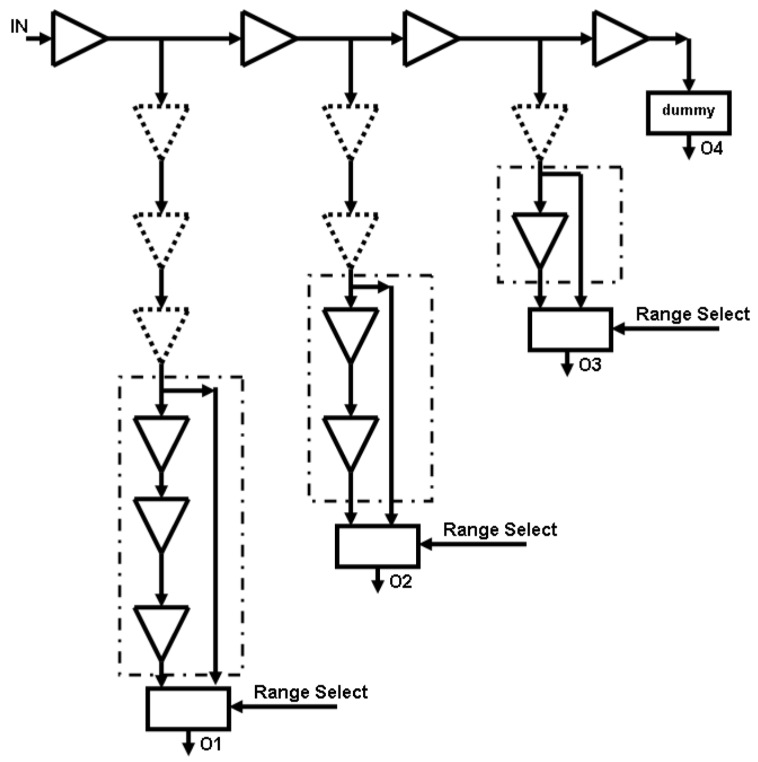


Figure 4.4: Beamforming delay subsystem

The delay cells indicated by solid lines are controlled by one set of digital code words and the delay cells in dotted lines are controlled by another set of code words. But these delay cells are identical. When both the delay elements have the same input code word, a relative delay of close to zero ps can be obtained in the design. When the delay of the two delay elements differ by 10ps then the relative delay between each path will be this delay difference of 10ps. The delay resolution that can be obtained depends on the minimum delay difference that can be obtained between the two delay elements. With this architecture, delay in the order of tens of picoseconds can be obtained which is a significant improvement compared to [1]. The beam forming delay tree can generate a delay difference in the range of 0-300ps. To double this range, additional set of delay elements as indicated within the dotted rectangular boxes in Figure 4.4 are used, to obtain the required delay range of 0-600ps. The range can be selected with the help of a range select input. In order to provide the accurate delay, the help of the digital calibration system with the fractional-N sigma-delta DLL and SAR DLL is required.

4.2.2 Fractional –N sigma-delta delay lock loop

The requirements of the true time-delay elements were listed in Chapter 2. From the requirements, it can be seen that to implement a beamforming UWB system, the timing jitter should be minimized. In [1], a digitally controlled PLL was used to control the delay line. But when compared to PLL, DLL has no cycle to cycle jitter. The DLL is more stable than PLL and easier to design [34].Hence, using a DLL instead of a PLL can improve the jitter performance. A DLL normally locks to one period of the reference clock. It can provide the required delay based on the reference clock period and the number of delay elements used. With a fixed frequency reference signal, the number of delay elements determine the phase resolution that can be obtained in a delay locked loop. A large number of delay cells are needed to obtain a very fine resolution.

Fractional-N PLLs were designed to obtain higher frequency resolution [18]. And sigma –delta fractional-N PLLs were used to randomize the division ratio to eliminate the spurious tones as well as achieving better frequency resolution[19]. Same concepts can also be applied to the DLL to improve the delay resolution. The architecture of the sigma-delta DLL used in this transmitter is shown in Figure 4.5.

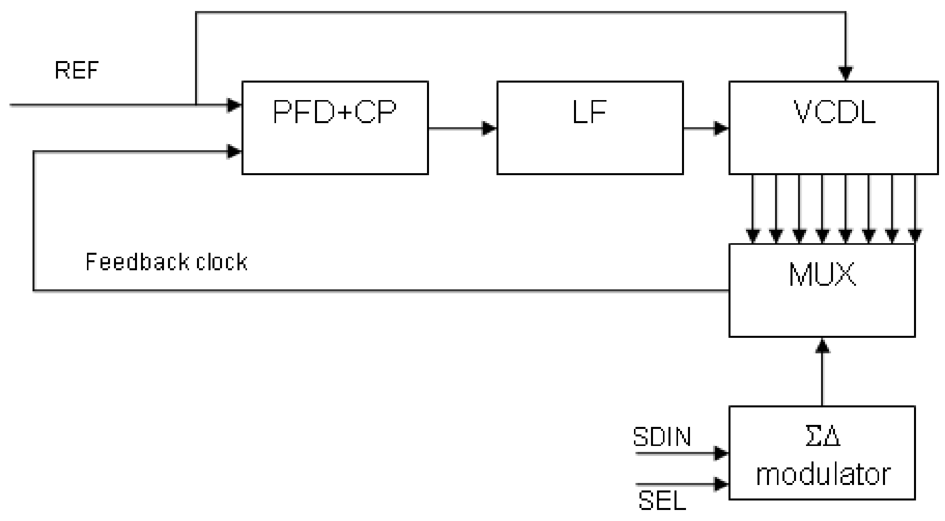


Figure 4.5: Fractional-N sigma-delta DLL architecture

A DLL normally would adjust the delay line to provide one cycle delay. The fractional-N sigma-delta DLL is similar to a normal DLL except that the sigma-delta modulator controls the feedback clock. The multi-phase clocks produced by the DLL are fed to a multiplexer that is controlled by the sigma-delta modulator. The output of this multiplexer is fed to the PFD. This DLL is based on the principle that the sigma-delta modulator controls the feedback clock from the VCDL and hence it controls the number of delay cells. The sigma-delta modulator would generate an average feedback clock controllable by the modulator input. Based on the average feedback clock generated, the DLL would settle to produce the delay. Therefore by controlling the average number of delay cells present in the DLL loop, the desired delay can be generated. The primary advantage of using the fractional-N sigma-delta delay locked loop is that it has the capability to generate a fractional delay without the need of additional phase interpolator. It also does not require a long delay chain to produce delay with very fine resolution. For example, with a reference clock frequency of 200 MHz and a delay line consisting of 20 delay cells, by using the output phases from delay cells 8 to 20, delay per cell in the range of 625ps to 250ps can be obtained respectively. By averaging the phases with a sigma-delta modulator that can generate 16 average phases, a theoretical delay resolution in the order of 5ps can be achieved.

4.2.3 Sub-block models of sigma-delta DLL

4.2.3.1 Voltage controlled delay line (VCDL)

The DLL consists of an analog voltage controlled delay line covering the range of delay required for beamforming. This work uses 20 delay cells to obtain the required delay range and resolution for beamforming. The delay cells are identical and the voltage from the loop filter controls the delay.

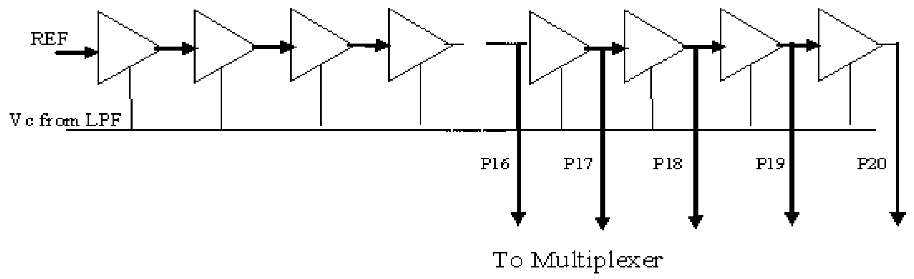


Figure 4.6: VCDL used in sigma-delta DLL.

The delay model in MATLAB simulations is assumed to have a nominal delay of 200ps and a gain of 1ns/volt. The nonlinearity of the delay line transfer characteristic does not affect the operation of the DLL. Hence, a simple current starved inverter delay is sufficient for the purpose.

4.2.3.2 Sigma-delta modulator

The DLL uses a second order sigma-delta modulator to produce the average feedback clock. The noise shaping provided by the sigma-delta is higher with higher order modulators. However, using a higher order modulator in the DLL would require a higher order loop filter which might render the loop unstable. Hence, a second order sigma-delta modulator with a 2-bit quantizer is used in the design which can switch between 4 phases of the clock signals to produce the average feedback clock.

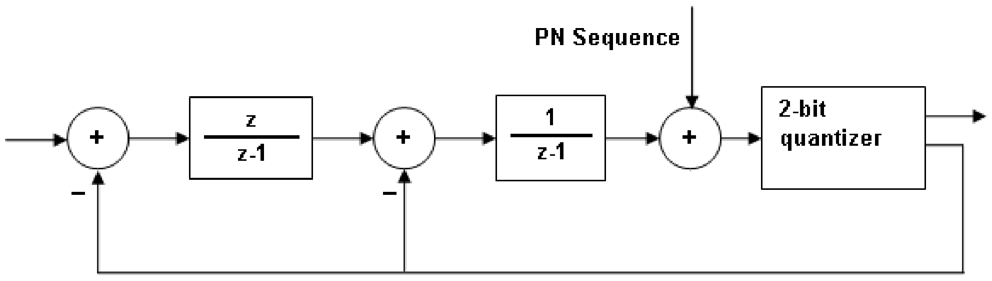


Figure 4.7: 2nd order sigma-delta modulator

The pseudo-random noise was added to dither the output of the sigma-delta modulator to eliminate the spurious tones. The presence of spurious tones would worsen the jitter performance. The simulated output spectrum of the sigma-delta modulator block is as shown in Figure 4.8.

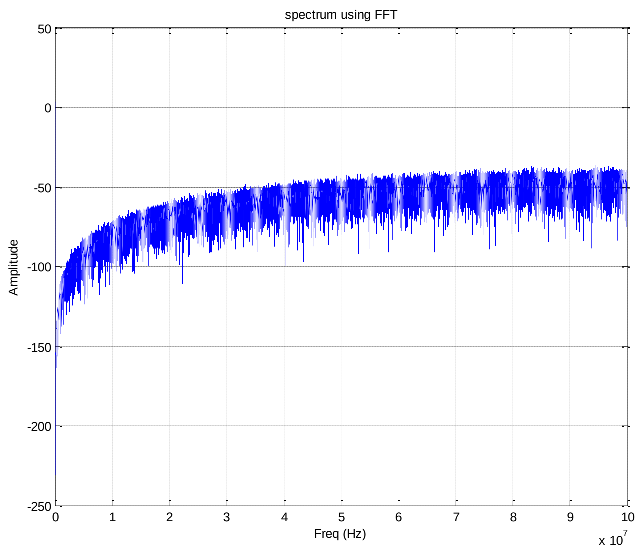


Figure 4.8: 2nd order sigma-delta modulator spectrum

The sigma-delta DLL shown in figure 4.5 was constructed in MATLAB. The sigma-delta modulator takes two inputs. The ‘SEL’ input is used to choose the group of clock phases for averaging. The ‘SDIN’ will set the average value. The ‘SDIN’ can take values in the range of -8 to 7 and the sigma-delta modulator can produce 16 different average phases by controlling the average number of delay cells in the DLL loop. The mapping between the inputs ‘SDIN’, ‘SEL’ and the average number of delay cells is given by

$$\textit{Avg. No. of delay cells (ND)} = 7+SEL+(8+SDIN)/16 \tag{4.1}$$

and the delay generated per delay cell is given by

$$\textit{Delay/cell (D)} = T_{REF}/ND \tag{4.2}$$

where, T_{REF} is the period of the reference clock and ND is the average number of delay cells. This design uses a reference clock with a period of 5ns. The delay corresponding to different modulator control words are shown in Table 4.1

Table 4.1: Sigma-delta delay lock loop – MATLAB simulation results

SDIN	**SEL**	**Avg. No. of delay cells**	**Expected delay(ps)**	**MATLAB simulation (ps)**
0	1	8.5	588.2	588.6
-5	2	9.1875	544.2	544.4
-8	4	11	454.5	454.8
-2	**5**	**12.375**	**404.0**	**404.6**
-4	7	14.25	350.8	351.0
-8	10	17	294.1	294.3

The results of the sigma-delta DLL obtained from MATLAB simulation for a particular input of -2, 5 are shown in Figure 4.9. This result depicts that the sigma-delta DLL settles down and produces a delay of 404.6ps per delay element corresponding to the input. This clearly shows that the fractional-N sigma-delta DLL architecture can generate fractional values of delay and the resolution no longer depends on the number of delay elements in the delay line.

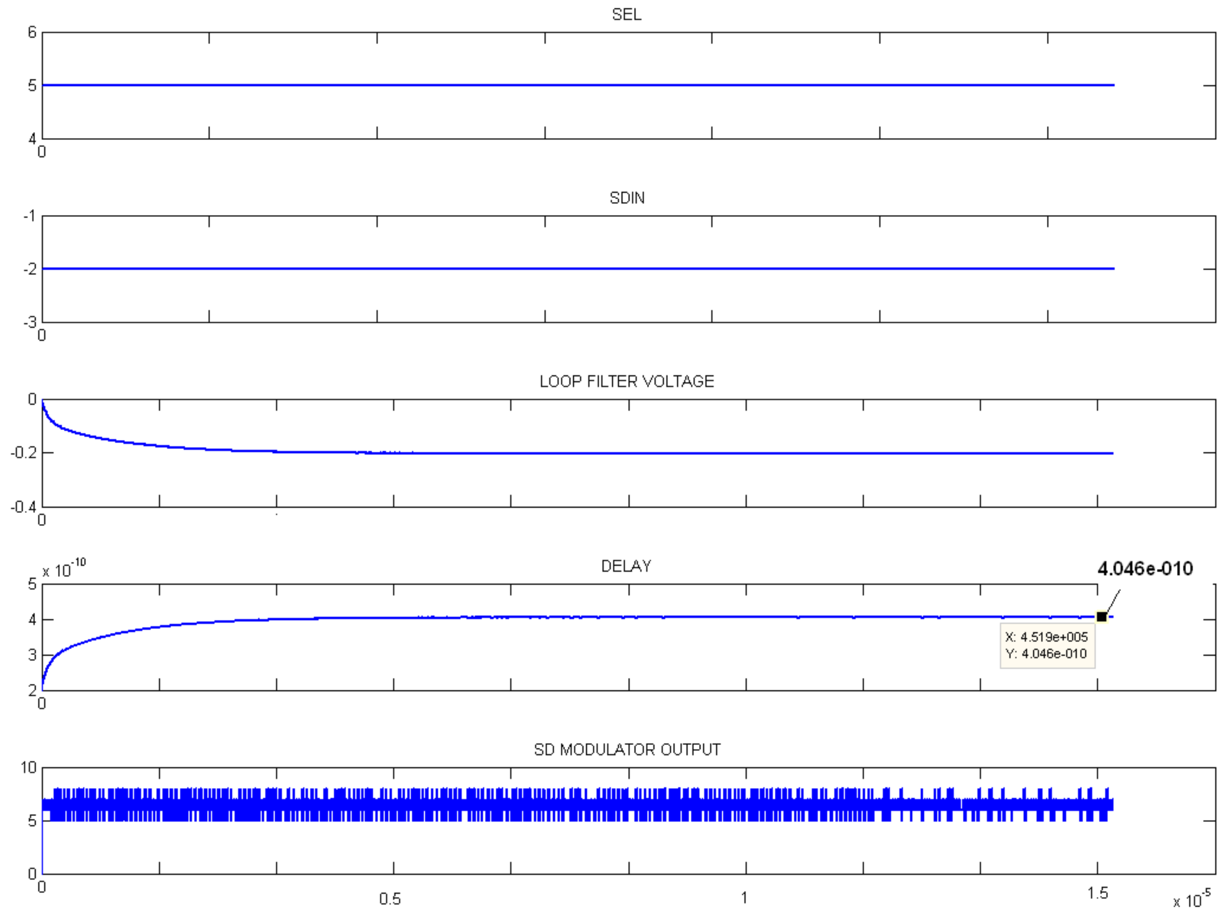


Figure 4.9: Fractional-N sigma-delta DLL settling and delay generation

4.2.4 SAR delay lock loop

Successive Approximation Register controlled (SAR) DLLs are very popular in digital delay lock loop architectures [21, 22]. A SAR digital delay lock loop uses a binary search algorithm to find the digital code word that would generate the required delay.

Using the sigma-delta DLL directly to generate the delay required for beamforming would require the delay lock loop to run continuously. This results in power consumption. To minimize the power consumption, a SAR delay lock loop is coupled with the sigma-delta delay lock loop to convert the analog delay into a digital code word.

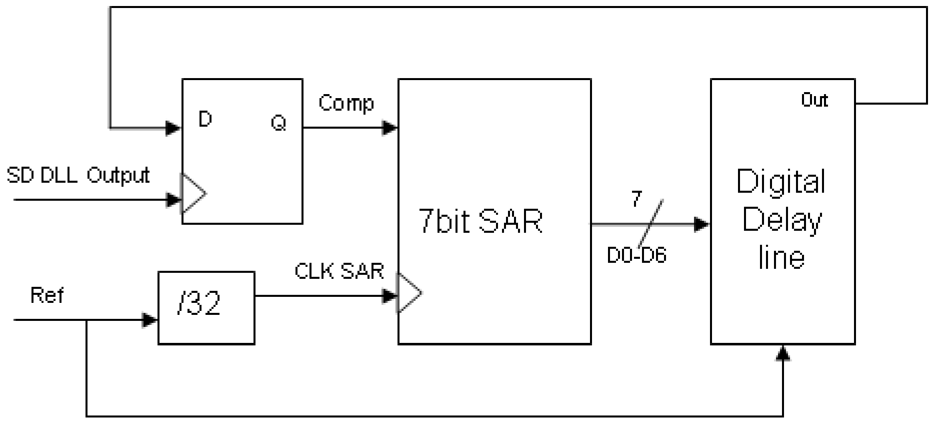


Figure 4.10: SAR DLL architecture

First, the sigma-delta delay lock loop will generate the desired precise delay for calibration. The SAR DLL compares the reference clock signal with the delayed signal obtained from sigma-delta DLL. The SAR will output the digital control word that will produce the delay that closely matches the delay produced by the sigma-delta DLL. The mismatch will be due to the limited resolution of the digital delay element employed in the SAR DLL. In order to minimize the mismatch, the digital delay element designed for the SAR DLL should have a very good delay resolution. The resulting control word can then be stored in the registers after calibration and used for controlling the beamforming transmitter. After calibration, both sigma-delta DLL and SAR DLL can be turned off. This will minimize the power consumption.

4.2.5 All-digital UWB transmitter block

The objective of this project was to develop a beamforming ultra-wideband transmitter. The beamforming is implemented in baseband as discussed in the previous sections. Now, an all-digital UWB transmitter that can form the UWB pulses and deliver to the antenna is required. There are several architectures which generate FCC compliant UWB pulses for transmission. Pulse shaping is one of the prominent techniques adopted to obtain an UWB pulse compliant to the FCC mask.

In [25], the transmitter generates an amplitude modulated UWB signal by multiplying baseband signal with an UWB monocycle train. The major drawback of this architecture is the leakage of the LO signal to the output which requires lowering of output power to meet the FCC mask. The architecture in [27] uses a phase lock loop with LC VCO and FIR filters to generate a FCC compliant UWB pulse. However, it is very power consuming and not suitable for burst mode operation due to long start-up delay in PLL. Tuned amplifiers with LC pulse shaping ladder can easily achieve spectral compliance but with significant area and power penalty [26]. The pulse shaping and power amplifier can also be implemented digitally by using a triangular pulse generator [28]. However, this architecture needs an external balun to form the UWB pulse.

The architecture in [2] is energy efficient and combines the edges from the output of a delay line to form the UWB pulse, but it needs an external filter to make the UWB pulse FCC compliant. The architecture in [3] is an all-digital architecture, which does not need any external filter or balun to form the UWB pulse and is energy efficient. The all-digital UWB transmitter implemented in this work uses a combination of both these architectures. This architecture consists of a digitally controlled delay line calibrated by the digital calibration subsystem. The UWB pulse is formed by combining the edges that are output from this delay line. Pulse shaping is done with help of a pulse shaping logic circuit that generates signals to control the pulse shape and a dual capacitively coupled digital power amplifier. The pulse shaping logic circuit used in this transmitter provides more flexibility to tune the pulse shape when compared to the logic implemented in [3]. The advantage of using this architecture is that it does not need any inductor or external balun to form the UWB pulses and hence is less area consuming, which is critical for a beamforming transmitter. The architecture used to implement the all-digital UWB transmitter block is shown in Figure 4.11.

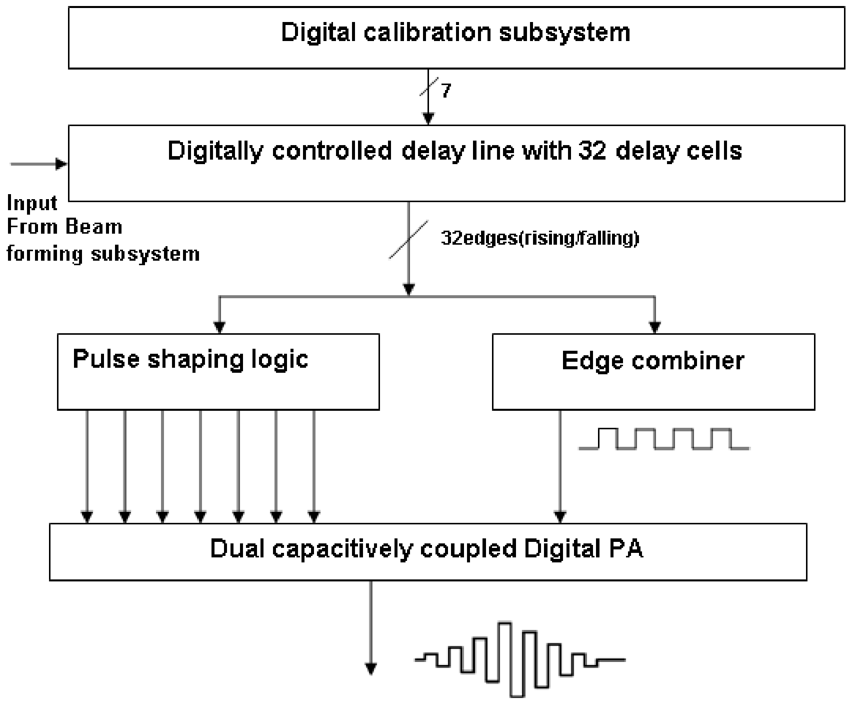

Digital calibration subsystem
7
Digitally controlled delay line with 32 delay cells
Input
From Beam
forming subsystem
32edges(rising/falling)
Pulse shaping logic
Edge combiner
Dual capacitively coupled Digital PA

Figure 4.11: All-digital UWB transmitter architecture

The reason for using the delay line and edge combiner to form UWB pulse instead of using the ring oscillator as in [3] is that the delay line can be accurately calibrated with the help of the sigma-delta delay lock loop. Hence, the centre frequency can be controlled precisely. Also, the DLL jitter is better than the ring oscillator jitter [34].

This architecture can control the number of pulses that should be formed per bit at the output. The shape of the pulses can be controlled with help of the pulse shaping logic and the power amplifier. The gain is also tunable giving us a flexibility to change all parameters of the output pulse to obtain the FCC compliant UWB pulse.

CHAPTER 5 : CMOS IMPLEMENTATION

The beamforming UWB transmitter was implemented in IBM 0.13µm CMOS process. This chapter gives the details of the implementation as well as the results obtained from Cadence simulations.

5.1 Digital delay element for beam former subsystem

The 7-bit digital delay element was implemented as a current starved inverter with digital PMOS switches (only 6-bit control is shown in Figure 5.1 for simplicity). The delay element was designed as per the guidelines indicated in [31].

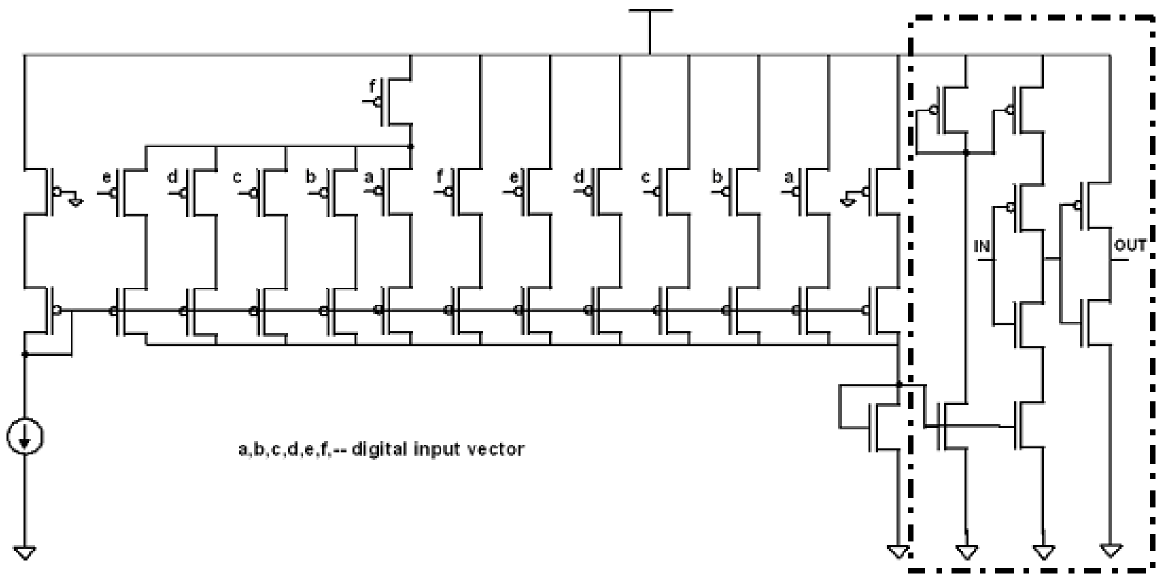


Figure 5.1: Digital delay element for beam former subsystem architecture (6-bits)

This architecture of using a current source to set the current for the delay element is more robust in terms of the temperature and process variations [31]. It is better than controlling the delay by just varying the size of the switches. The resolution of the digital delay element determines the achievable resolution.

Since fractional–N sigma-delta delay locked loop can generate a delay with precise delay resolution, the bottleneck is the resolution provided by the digital delay element. Resolution in the order of 5-10ps is required for the current system. The current starved delay element has a nonlinear characteristic with respect to the current. Hence, resolution of the order of 5-10ps is not obtainable at low currents. The curve is as shown in Figure 5.2.

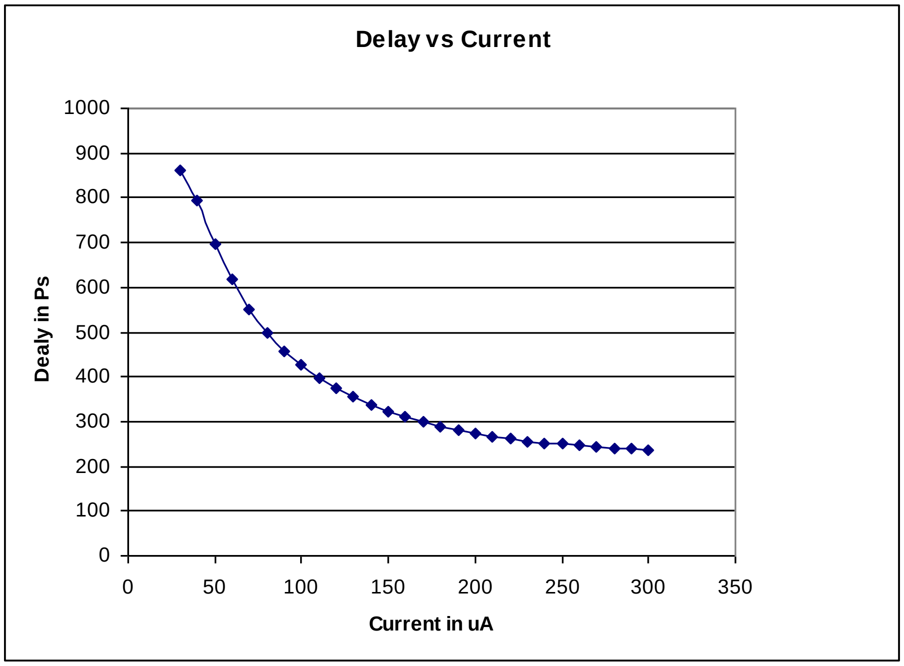


Figure 5.2: Delay vs. Current in current starved inverters

In order to overcome this nonlinearity and obtain a more linear curve which can provide a delay resolution of about 5-10ps, additional transistors were added to make the characteristics more linear.

The following procedure was adopted to obtain the sizing for the transistors. The required delay range for the digital delay element is from 300ps to 600ps. Taking into account the process variations; the delay element was designed in the typical corner with a margin to satisfy a range of 200ps to 790ps. This margin helps to obtain the required delay range in all process corners. A resolution of 10ps is needed for the beam forming system to steer the beam by 1 degree. To achieve this, a resolution of the order of 5ps is needed from the digital delay element. Hence, the number of bits required to cover the delay range of 200 to 790ps in typical corner with a resolution of 5ps is determined to be 7 bits.

With 7 bits of control, there are 128 combinations for the input. The delay range to be achieved was broken down into four groups. The maximum and minimum delay for each range was calculated based on the total maximum delay and the resolution to be obtained. The 5 bits controlling each range was then designed based on the procedure in [31]. Then the transistors controlling the four groups were merged appropriately to obtain the complete 7-bit controlled digital delay element. The resulting delay element has a delay characteristic that is more linear than the conventional characteristics.

This linearization of the delay characteristics was done by just adding a few more transistors. These transistors need to be added only for the biasing circuit that would generate the biasing voltage for the current starved inverters. They need not be replicated in each delay cell. The delay cells will consist of just the current starved inverter section with the current mirror section as indicated in the Figure 5.1 by the rectangular dotted box. This linearization guarantees a resolution of 5-10ps as required by the system. The delay line was simulated in various temperatures and corners, and it covers the range of 300-600ps in all temperatures and corners. The result is shown in Figure 5.3.

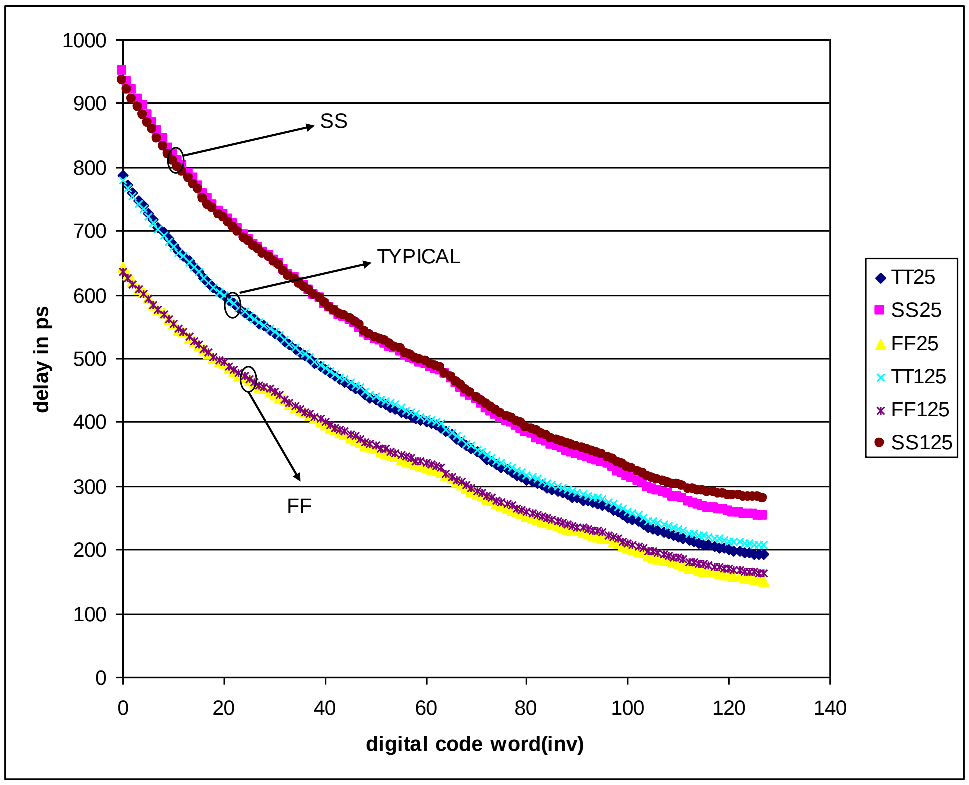


Figure 5.3: Cadence corners/temperatures simulation of digital delay element

This delay cell will also be used in SAR DLL for beamforming delay calibration.

5.2 Fractional –N sigma-delta delay lock loop

5.2.1 Phase frequency detector and startup circuit

In the sigma-delta delay locked loop, a phase frequency detector is used to compare the phases between the reference clock and the output feedback clock. The phase frequency detector implemented is a conventional phase frequency detector consisting of two D flip-flops and a nand gate as shown in Figure 5.4.

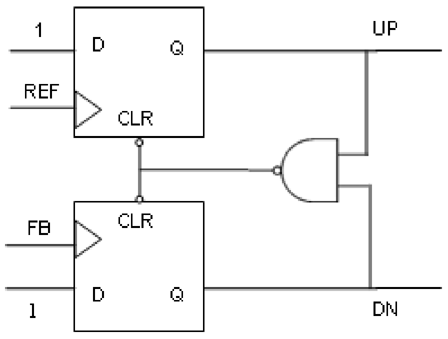


Figure 5.4: Conventional phase frequency detector

The phase frequency detector (PFD) generates the UP and DN signals based on the phase difference between the reference clock and the feedback clock. When both the reference clock and feedback clock are in phase, the UP and DN signals generated from the PFD will have the same pulse width. This is the smallest pulse width that these signals can have. This smallest pulse width has been designed to be around 250ps to avoid dead-zone problem. This normal phase frequency detector suffers from the false lock problem because of the limited initial condition. Hence, a start controlled circuit is necessary to set the initial condition of the phase frequency detector [29].

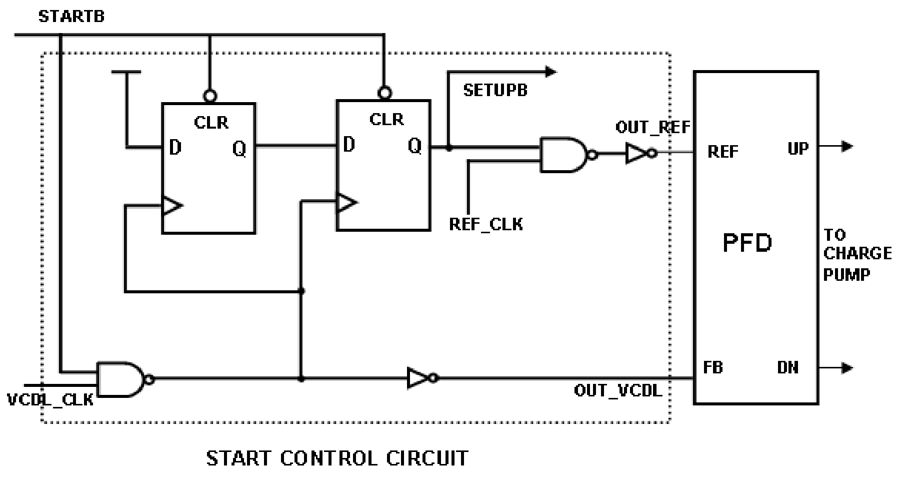


Figure 5.5: PFD and start-controlled circuit

The start controlled circuit was adopted from [29]. The addition of this start control circuit sets the initial condition. When the ‘startb’ signal is low the two clocks are masked off. Hence, the PFD is disabled. When the ‘startb’ signal goes high the two clocks propagate to the PFD for comparison. Also, the ‘setupb’ signal is used to pull the initial filter voltage to VDD through a PMOS transistor. This sets the initial delay at the minimum value, since higher control voltage gives lower delay.

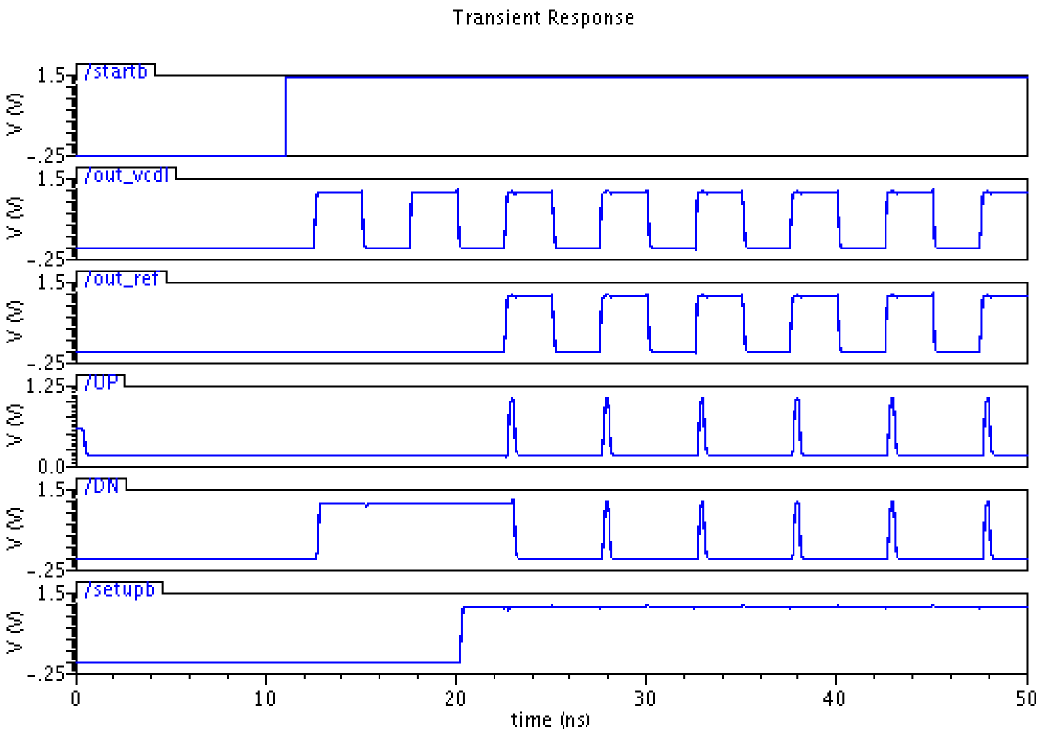


Figure 5.6: Simulation of start controlled circuit with PFD

The simulation result of the start controlled circuit with the PFD is indicated in Figure 5.6. The start controlled circuit avoids the false lock problem that would exist otherwise.

5.2.2 Charge pump

The charge pump circuit converts the phase difference to a current. Based on the UP and DN signals from the phase detector, the charge pump circuit will provide the current to charge or discharge the loop filter and thereby generating the control voltage V_{CTRL}. The charge pump circuit used here is adopted from [29] and its schematic is shown below.

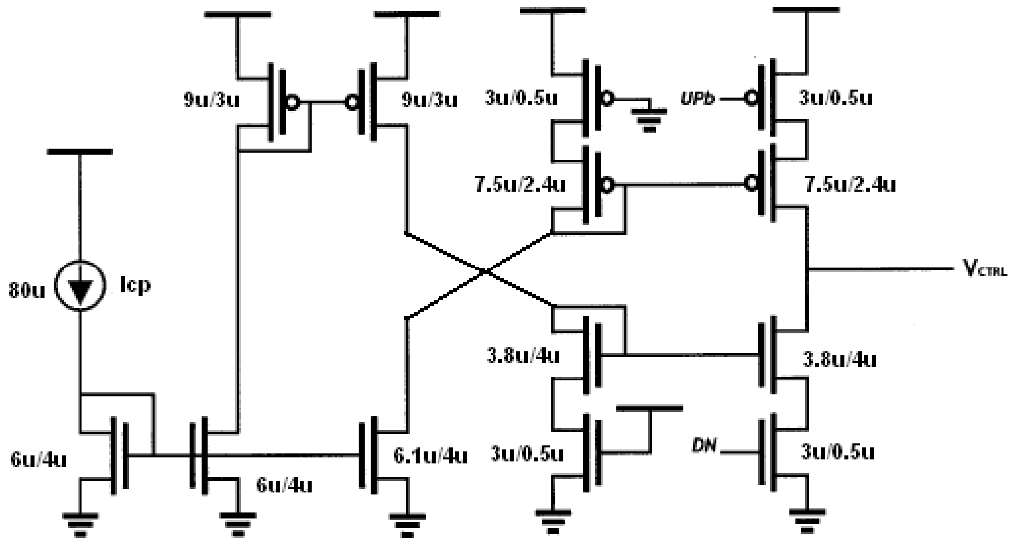


Figure 5.7: Schematic of charge pump adopted from [29].

The charge pump uses long length transistors to obtain good mirroring of current. The switches controlled by the UP and DN signals uses smaller length transistors (0.5µm) to provide faster switching with lesser parasitic capacitances. This charge pump design mitigates charge injection errors induced by the parasitic capacitance of the switches and current source transistors [30]. The charge pump implemented in the design uses a current of 80µA.

5.2.3 Loop filter

The loop filter used is a passive second order filter network consisting of a resistor and 2 capacitors as indicated in Figure 5.8.

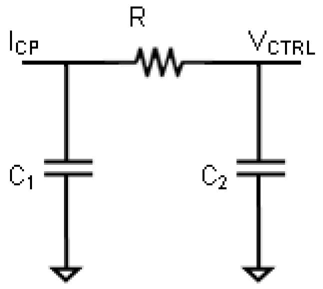


Figure 5.8: Loop filter

The transfer function of the loop filter is given by

$$F(s) = \frac{V_{CTRL}}{I_{CP}} = \frac{1}{s^2(C_1C_2R) + s(C_1 + C_2)} \tag{5.1}$$

The transfer function of a DLL is given in [30] as

$$D_O(s) = (D_I(s) - D_O(s))F_{REF}K_{VCDL}I_{CP}F(s) \tag{5.2}$$

Substituting the filter transfer function in the above equation and simplifying produces

$$\frac{D_O(s)}{D_I(s)} = \frac{F_{REF}K_{VCDL}I_{CP}}{s^2(C_1C_2R) + s(C_1 + C_2) + (F_{REF}K_{VCDL}I_{CP})} \tag{5.3}$$

The above equation can be simplified by assuming that the second pole is far apart from the first pole. The two poles of the equation are given by

$$\omega_{P1} = \frac{F_{REF}K_{VCDL}I_{CP}}{C_1} \tag{5.4}$$

and

$$\omega_{P2} = \frac{1}{RC_2} \tag{5.5}$$

The first pole is also called the loop bandwidth. The loop bandwidth of the DLL determines the jitter performance of the DLL. To obtain less jitter, the loop band width should be small. Normally a loop bandwidth of the order of one-tenth of the reference frequency is used. This design has a charge pump current of 80µA, a reference frequency of the 200MHz is used and the K_{VCDL} gain of the analog delay element is approximately 1.7ns/V.Hence with a capacitor (C_1) of 160pF, the loop bandwidth is given by

$$f_{BW} = \frac{\omega_{P1}}{2\pi} = 27kHz \tag{5.6}$$

Such a low value of loop bandwidth is selected to obtain a low value of jitter. The capacitor C_2 is chosen to be one-tenth of C_1 and the resistor is chosen to keep the second pole far from the first pole. The value of the resistor used in the design is 12kΩ. This gives a second pole frequency of 828kHz.

5.2.4 Voltage controlled delay line

The voltage controlled delay line (VCDL) is one of the important blocks of a DLL. The performance of the VCDL is very critical as it directly affects the jitter performance of the DLL. Since a sigma-delta DLL is a feedback loop, it will always settle to give the required delay. The non-linearity in the characteristics of the delay element does not affect the performance of the DLL greatly. Hence, the analog voltage controlled delay line required in the sigma-delta DLL was designed using voltage controlled current starved inverters as shown in Figure 5.9.

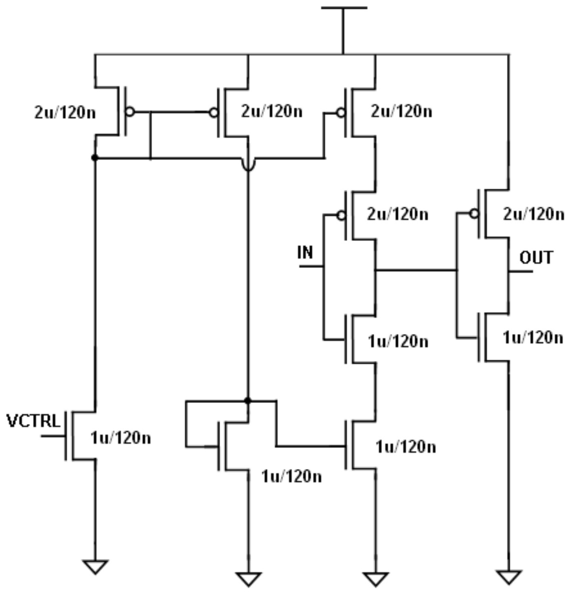


Figure 5.9: Schematic of voltage controlled delay element

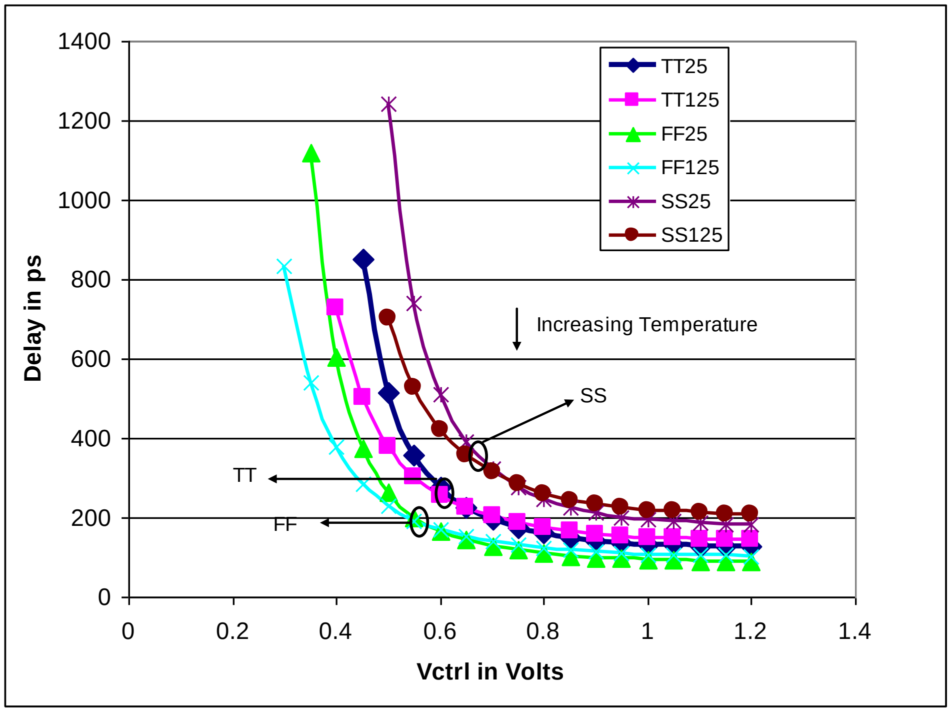


Figure 5.10: Cadence corners/temperatures simulation of voltage controlled delay cell

The delay cells have been sized to obtain the required delay range of 300-600ps. The delay element was simulated in various process corners and temperatures. As illustrated by the Figure 5.10, this delay cell is able to cover the required delay range. Hence, the sigma-delta DLL can provide the required delay in spite of PVT variations.

5.2.5 Sigma-delta modulator

The second order digital sigma-delta modulator was built using verilog. The adders needed to implement the sigma-delta modulator have been designed as carry look-ahead adders. Carry look-ahead adders operate very fast and provide the capability for the sigma-delta modulator to operate at very high frequencies. In the current design the sigma-delta modulator operates at a frequency of 200MHz. The pseudo-random noise generator used to randomize the sigma-delta modulator output is designed as a 25-bit linear feedback shift register that implements the generator polynomial ($z^{25}+z^{22}+1$). The verilog code of the sigma-delta modulator was used in Cadence mixed signal simulations.

5.3 SAR delay lock loop

The SAR delay lock loop is a digital delay lock loop that converts the analog delay into a digital code word. The SAR DLL consists of a SAR controller and a digital delay line. This design uses two SAR DLLs namely, the beamforming SAR DLL and the transmitter SAR DLL. The digital delay line in the beamforming SAR DLL consists of 5 delay cells that resemble the delay cells used in the beam former subsystem. The digital delay line in transmitter SAR DLL is made up of 16 delay cells that replicate the transmitter delay line. During the calibration of the beam former subsystem and the transmitter, the respective SAR DLL is used. A simple D flip-flop serves as phase comparator for the DLL. This design uses a 7-bit SAR controller to determine the digital code word corresponding to the delay generated by the sigma-delta DLL. The SAR controller is shown below [21].

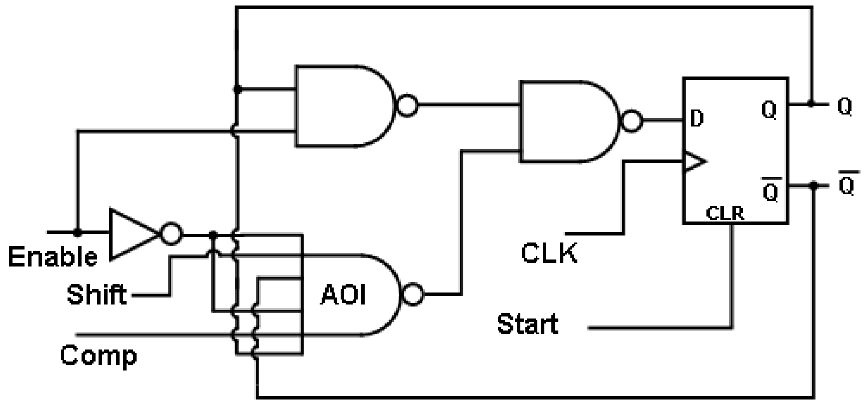


Figure 5.11: SAR controller

The output of 5th delay cell of the sigma-delta DLL is fed to the SAR DLL. The SAR controller was custom designed in Cadence. The 7-bit SAR controller will control the digital delay line. It will try and produce the closest matched delay as generated by the sigma-delta delay lock loop and obtain the corresponding digital code word.

5.4 Digital calibration circuit

The digital calibration FSM was built in verilog. The FSM has registers that can be programmed through the serial to parallel interface. Based on the programming, the FSM will calibrate the beamforming delay subsystem and it will also calibrate the transmitter delay cell to produce the required centre frequency. The FSM implemented in verilog is depicted in Figure 5.12.

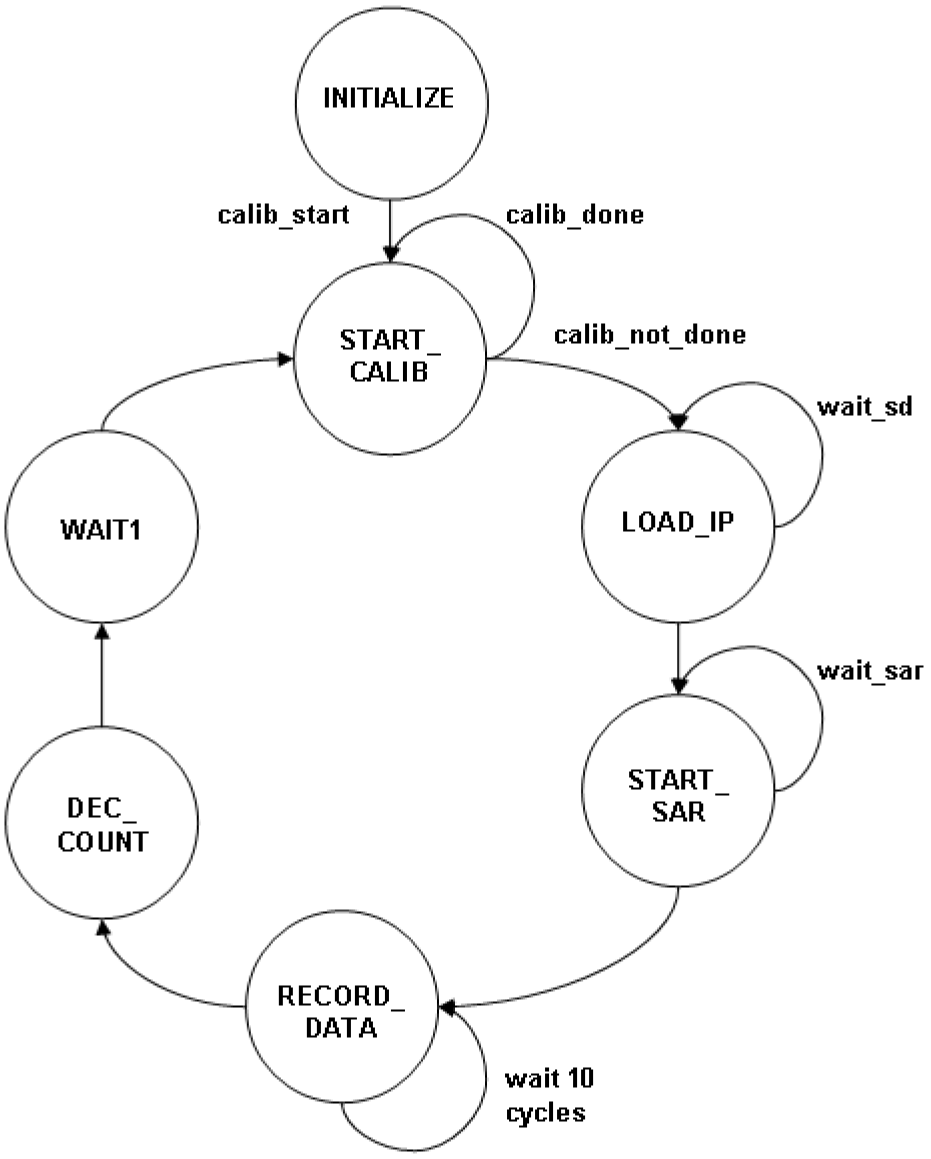


Figure 5.12: Digital calibration FSM

The FSM consists of 7 states. The descriptions of these states are as follows.

INITIALIZE: The FSM enters this state as soon as the system is reset. It remains in the same state until the calib_start bit is set.

START_CALIB: After programming the required calibration configuration and providing the inputs needed for calibration, the calib_start bit is set and the FSM reaches this state. If the calibration is complete, then the FSM stays in the same state. If calibration is not complete, then the FSM checks the calibration that needs to be done and proceeds to the LOAD_IP state. The FSM also resets the sigma-delta DLL and SAR DLL in this state.

LOAD_IP: This is the load input state. In this state the inputs needed for generating the required delay are fetched from the registers and the FSM provides them to the sigma-delta DLL and activates the latter. The FSM remains in the same state until the sigma-delta DLL settles. After the sigma-delta DLL settles the FSM proceeds to the START_SAR state.

START_SAR: This state activates the SAR DLL and waits for the SAR DLL to lock. Once it locks, the FSM obtains the digital code word corresponding to the required delay. It then moves to the RECORD_DATA state.

RECORD_DATA: This state records the digital code word obtained for the required delay into the register files. It waits about 10 cycles before proceeding to the next state.

DEC_COUNT: This state decrements the calibration counter for every set of input that has been calibrated by the FSM. The calibration becomes complete when the calibration counter reaches zero.

WAIT1: The FSM waits for one clock cycle in this state before proceeding to the START_CALIB state. This additional wait is to give sufficient time for the decrement operation to happen.

The inputs to the sigma-delta DLL control the beam former subsystem delay and the transmitter centre frequency. By programming the input, the centre frequency can be selected to be 3.5 GHz, 4 GHz or 4.5 GHz. The delay difference of the two delay elements used in the beam former subsystem can be controlled and hence the beam steering angle can be controlled. The calibration system feeds the digital code words to the beam forming delay subsystem and the transmitter. These in turn, generate four UWB signals with the desired centre frequency each separated by the required delay difference. Inputs corresponding to different delay values and three centre frequencies can be provided to the calibration FSM. From the results of the calibration obtained, the delay difference and the centre frequency to be used can be chosen. The complete digital calibration algorithm is summarized with help of a flowchart shown in Figure 5.13.

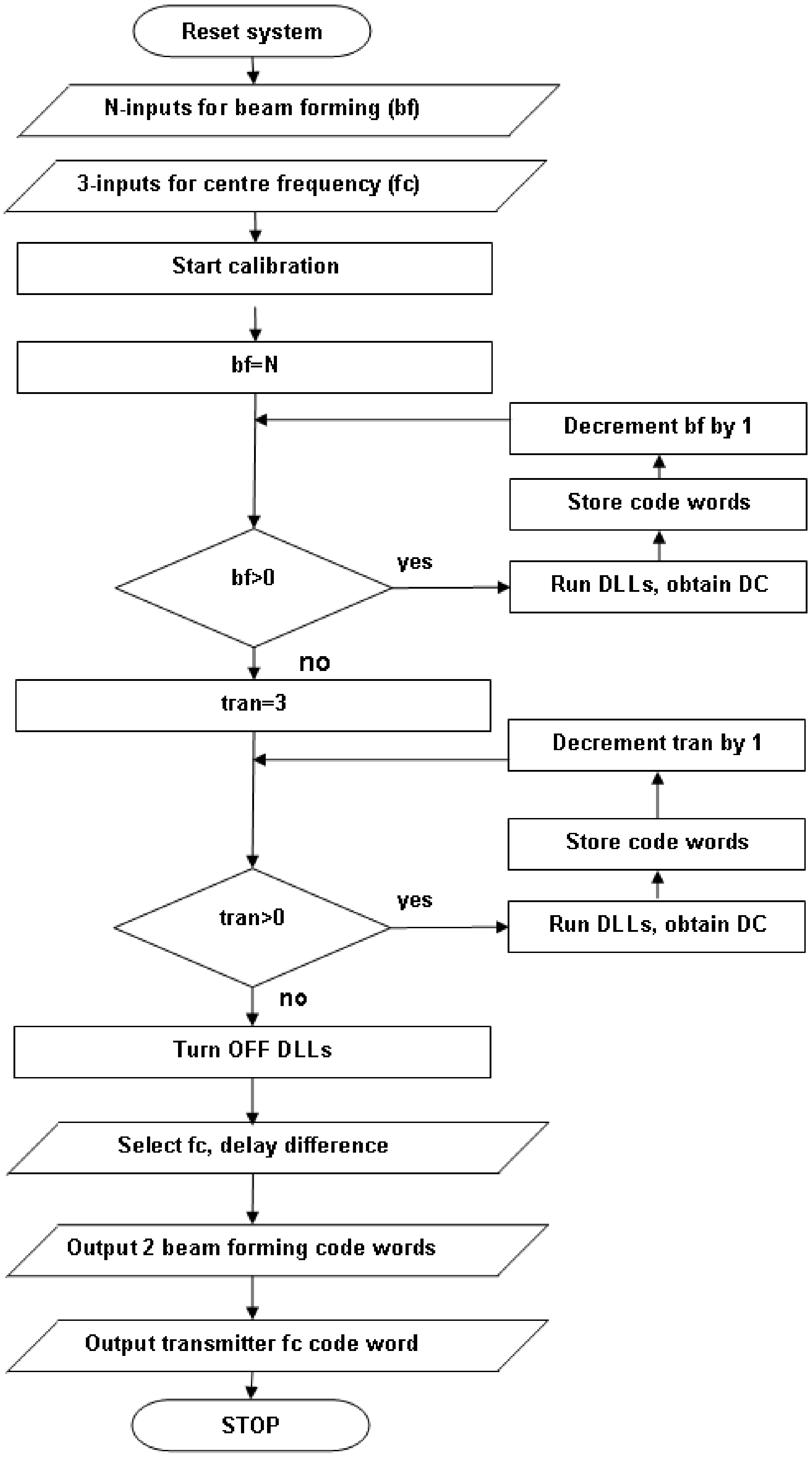


Figure 5.13: Digital calibration algorithm

The serial to parallel interface and the calibration algorithm were developed in verilog and they were implemented in FPGA and tested. The verilog models were also ported to Cadence. The digital calibration algorithm was verified in Cadence SpectreVerilog mixed signal simulation and was able to provide the required transmitter delay and beamforming delay code words. The sigma-delta DLL generates a delay in the range of 300ps to 600ps per delay cell, but the transmitter delay cell generates a delay in the range of 90 to 150ps. Hence, to utilize the same delay range generated by 5 delay cells of the sigma-delta DLL to calibrate the transmitter centre frequency, the SAR DLL compares the analog delay with the delay generated by 16 delay cells of the transmitter SAR delay line. The relation between the sigma-delta DLL inputs and the expected transmitter delay per cell can be obtained by

$$\textit{Transmitter delay/cell} = 5(\textit{Delay/cell})/16 \qquad (5.6)$$

where, *Delay/cell* is given by equation 4.2.

The results obtained from Cadence simulation for the transmitter centre frequency calibration are tabulated in Table 5.1.

Table 5.1: Transmitter delay centre frequency calibration from Cadence

SEL	SDIN	Expected centre frequency (GHz)	Expected delay in ps	Transmitter Delay Digital Codeword	Obtained delay(ps)	Obtained centre frequency (GHz)
3	6	3.5	143.68	1011111	140	3.57
5	-2	4.0	126.26	1010010	123	4.06
7	-8	4.5	111.61	1001100	109	4.59

These results demonstrate the capability of the transmitter to generate the precise centre frequency. The results for the beamforming delay obtained for different modulator control words are as follows

Table 5.2: Beamforming delay calibration results from Cadence

SEL	**SDIN**	**Avg. No of delay cells**	**Exp. delay/cell in ps**	**Obtained Delay/cell in ps**	**Digital Codeword**
2	-6	9.125	547.95	540	1010001
2	-4	9.250	540.54	530	1001111
5	**-2**	**12.375**	**404.04**	**398**	**0110100**
5	0	12.500	400.00	392	0110011
10	-8	17.000	294.12	292	0010111
10	-5	17.1875	290.91	287	0010011

The Cadence simulation result obtained for a particular input of 5, -2 is depicted in Figure 5.14. The obtained delay between the reference signal and sigma-delta DLL output is 2.007ns. This is the delay of 5 delay cells. Hence, the delay of one delay cell produced by sigma-delta DLL is 401.4ps. The delay between reference signal and the output of SAR DLL is 1.99ns. Hence, the delay of one delay cell generated by the SAR DLL is 398ps which is very close to the required delay of 404ps differing by just the digital delay resolution.

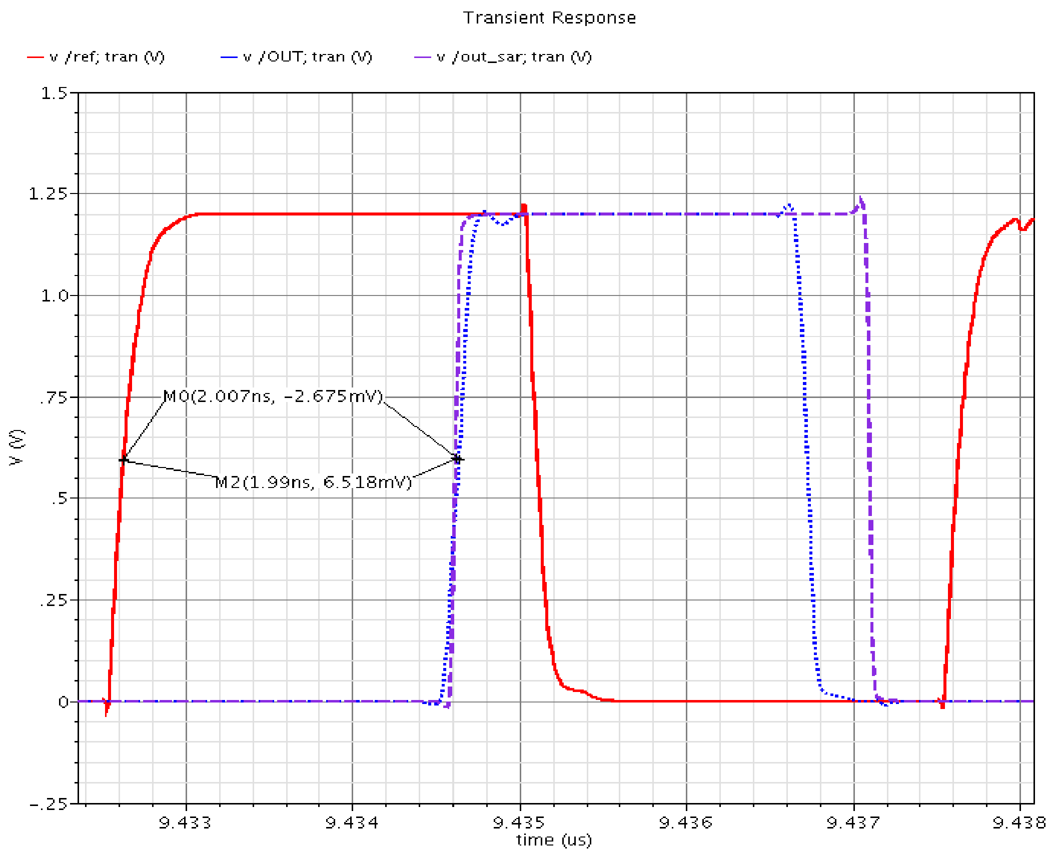


Figure 5.14: Fractional-N sigma-delta & SAR calibration result from Cadence

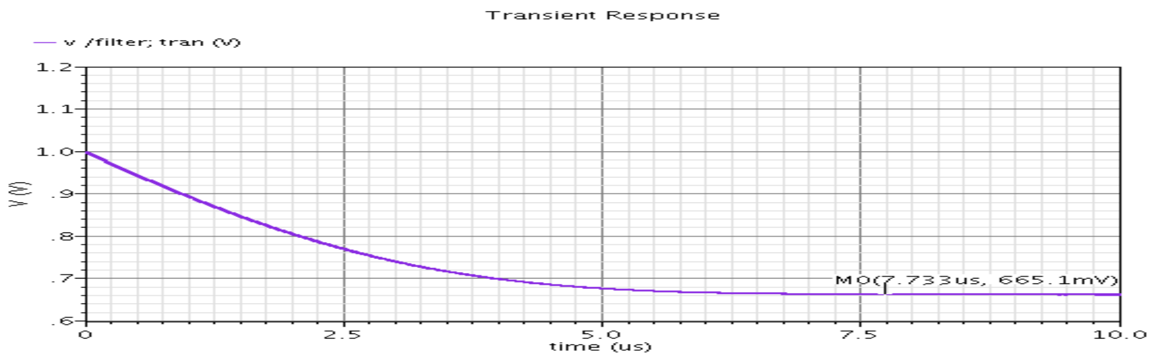


Figure 5.15: Fractional-N sigma-delta –settling of control voltage

The settling of the loop filter control voltage is shown in Figure 5.15. In order to speed up the simulation, the filter voltage was set at 1V instead of 1.2V.

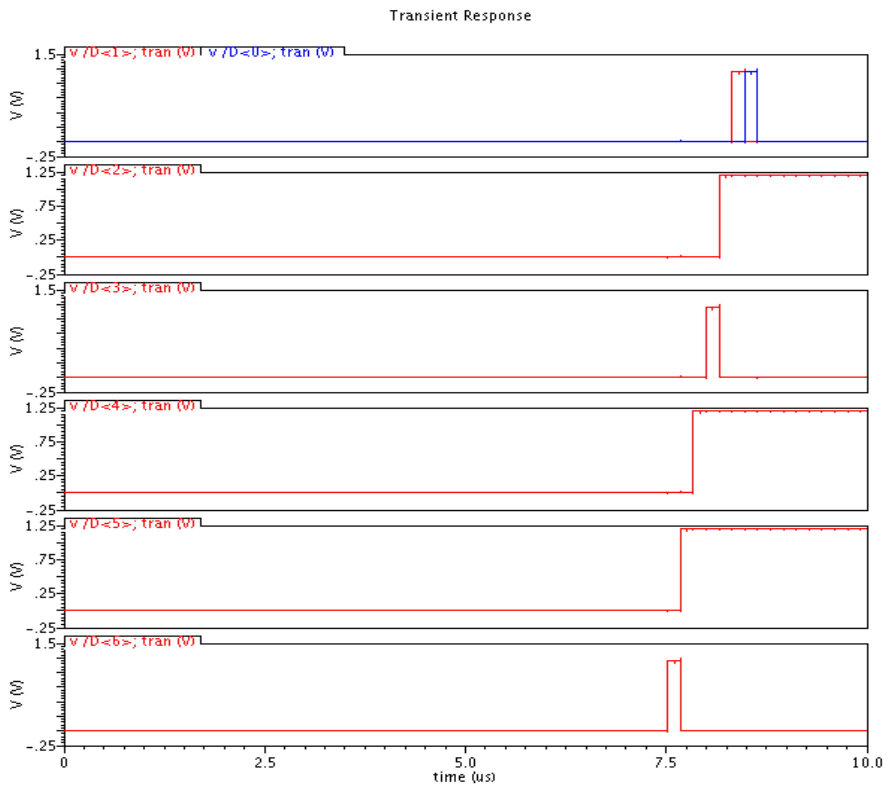


Figure 5.16: Digital code word from SAR DLL (MSB bottom -LSB top)

The figure 5.16 illustrates the digital code word obtained from the SAR DLL. Thus, the obtained digital code word for the required delay is 0110100. This digital code word is stored by the FSM.

The inputs to the sigma-delta DLL were varied to generate all possible delay values that can be generated by the system within the range of 300ps to 600ps. The delay difference between the different delay values obtained from the sigma-delta DLL and SAR DLL for adjacent inputs is plotted against the delay in Figure 5.17

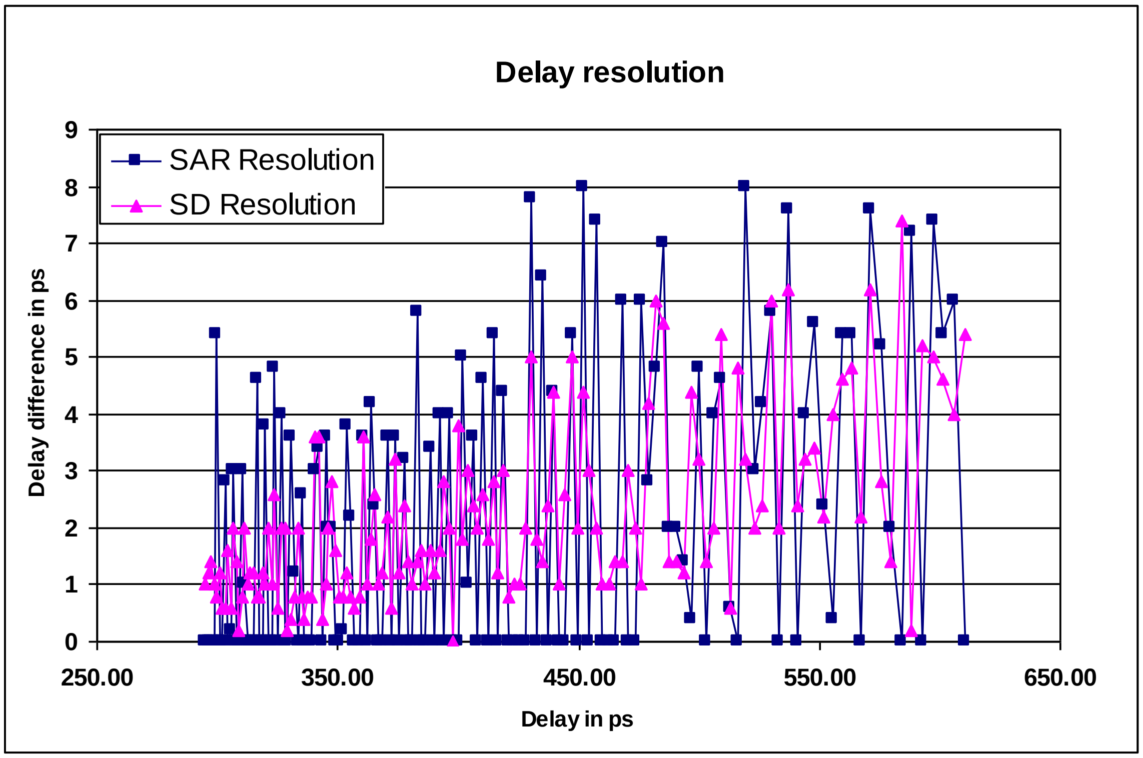


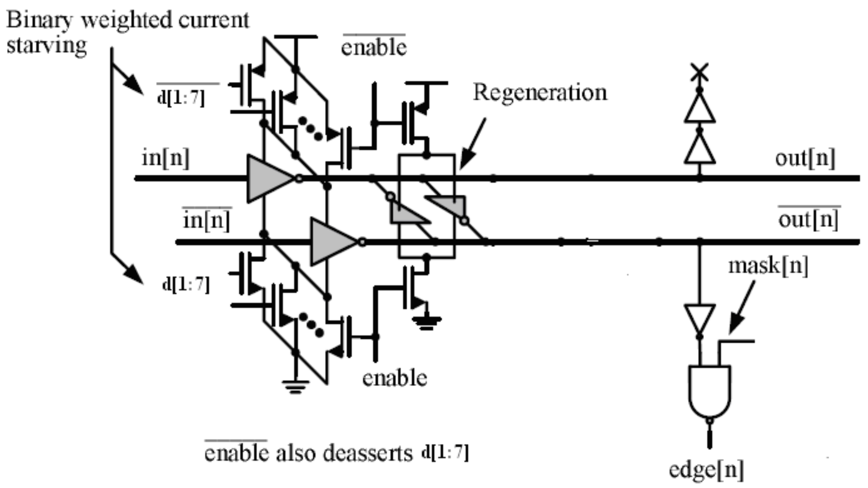
Figure 5.17: Delay resolution of Sigma-delta and SAR DLL

This plot clearly depicts the capability of the system to generate very fine delay. From the plot, it can be observed that both the sigma-delta DLL and the SAR DLL can generate a delay resolution of less than 10ps. Hence, the beam former delay subsystem with the vernier delay architecture can provide a delay resolution of less than 10ps.

5.5 All-digital UWB transmitter

5.5.1 Transmitter delay line

The digital transmitter consists of 7-bit digitally controlled pseudo differential current starved inverter delay line. This is based on reference [2]. This delay line uses only the sizing of the switches to control the delay. The switches are binary weighted.



Binary weighted current starving
$\overline{\text{enable}}$
$\overline{\text{d[1:7]}}$
Regeneration
in[n]
out[n]
$\overline{\text{in[n]}}$
$\overline{\text{out[n]}}$
mask[n]
d[1:7]
enable
$\overline{\text{enable}}$ also deasserts d[1:7]
edge[n]

Figure 5.18: 7-bit digitally controlled transmitter delay from [2]

No current source is used in this delay line in order to avoid static power consumption in the transmitter. The delay cell was designed to provide a delay in the range of 90ps to 150ps in all temperatures and corners. The cross coupled inverters connected at the output helps in regeneration and are used to equalize the rise and fall time of the signals [2]. The edges from the delay line are masked to control the pulse width of the output. With a 32-stage delay line, the pulse width can be tuned from 500ps to 4ns.

5.5.2 Edge combiner

The state preserving edge combiner was adopted from [12]. This edge combiner can work up to 5GHz. The edge combiner is shown in Figure 5.19.

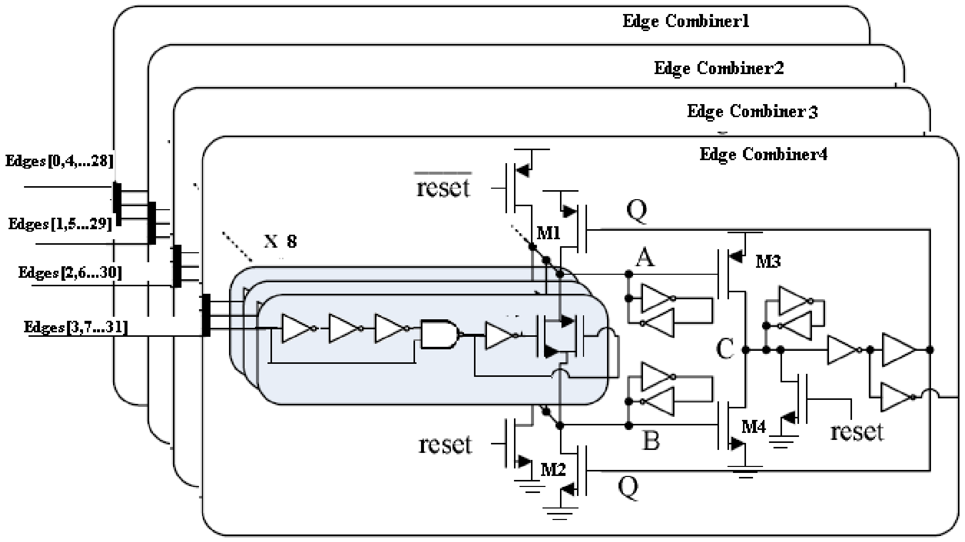


Figure 5.19: Time interleaved edge combiner from [12]

After reset, the node A is charged to VDD and node B is discharged to ground. The node C is discharged to ground and the NMOS transistor M2 turns on. When a positive edge comes, it creates pulse of width equal to three inverter delays. This pulse will turn on the transmission gate for a short period and the node A will now discharge through the transmission gate and the NMOS transistor M2 to ground. Once the node A voltage reaches VDD-Vtp of the PMOS transistor, M3 turns on and the output node C is charged up and this will change the value stored by the latch at C to high. Once C becomes high, Q becomes low. It will turn on the PMOS transistor M1 and turn off the NMOS transistor M2, and node A is charged back to VDD. When the next edge comes, the transmission gate is on and the node B is charged through the transmission gate and the PMOS M1 to VDD. Once the node B voltage crosses the Vtn of the NMOS transistor, M4 turns on. This pulls the node C to ground and changes the value of the latch at node C to low and hence Q becomes high. Now the PMOS transistor M1 turns off and the NMOS transistor M2 turns on, and the same process repeats itself for every positive edge fed to the combiner. The delay of the transmitter delay cell decides the centre frequency of the transmitter. The edge combiner combines these delayed edges to produce the UWB pulse.

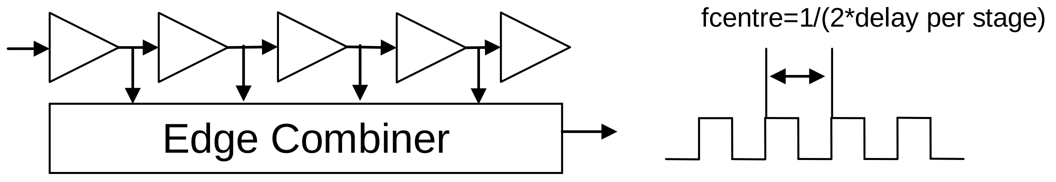


Figure 5.20: UWB pulse generation

Each edge combiner cell combines 8 edges to produce a pulse train. These pulses are then passed through a xor tree to produce the desired UWB signal.

5.5.3 Digital PA and pulse shaping

One of the power hungry blocks in a UWB transmitter is the power amplifier. Hence, an energy efficient power amplifier is needed. In [3], they employ a digital power amplifier (PA) using capacitively coupled pulse shaper for BPSK signal generation. The architecture is all-digital and is energy efficient. In their architecture, they use ring oscillator to generate the RF pulses. A pulse shaping circuitry is used to provide pulses to the digital PA to shape the output pulse. It uses just two capacitors to generate the required UWB pulse and is thus area efficient. Therefore, the same architecture is adopted with modifications to the pulse shaping logic to give more freedom in controlling the shape of the generated UWB pulse.

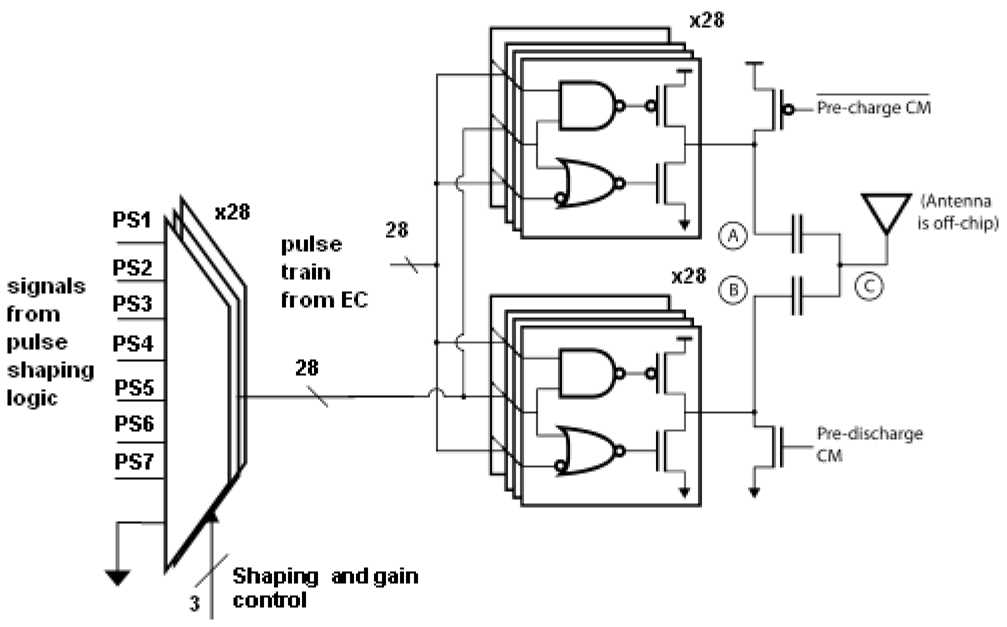


Figure 5.21: Dual capacitively coupled digital PA from [3]

The pulse shaping logic implemented in this design generates 7 pulse shaping signals. In comparison to the pulse shaping logic implemented in [3], the pulse shaping logic used in this design provides additional flexibility to control the amplitude of each pulse individually. This modification was primarily done, because the UWB signal generated with dual capacitively coupled digital power amplifier would result in an asymmetric UWB pulse signal. This asymmetry is due to the change in average DC value of the capacitors during pulse generation. The individual amplitude control of each pulse of the output UWB signal helps in overcoming this issue. It also provides more flexibility in obtaining a FCC compliant UWB pulse.

The basic concept on which the capacitively coupled pulse shaper [3] works is that when two pulses with in-phase RF components and out of phase common mode components are provided to the capacitors, the common mode components cancel each other and the

RF components propagate to the output. The digital PA circuits consist of tri-state inverters and pulse shaping is obtained by controlling number of inverters that are ON at a particular time.

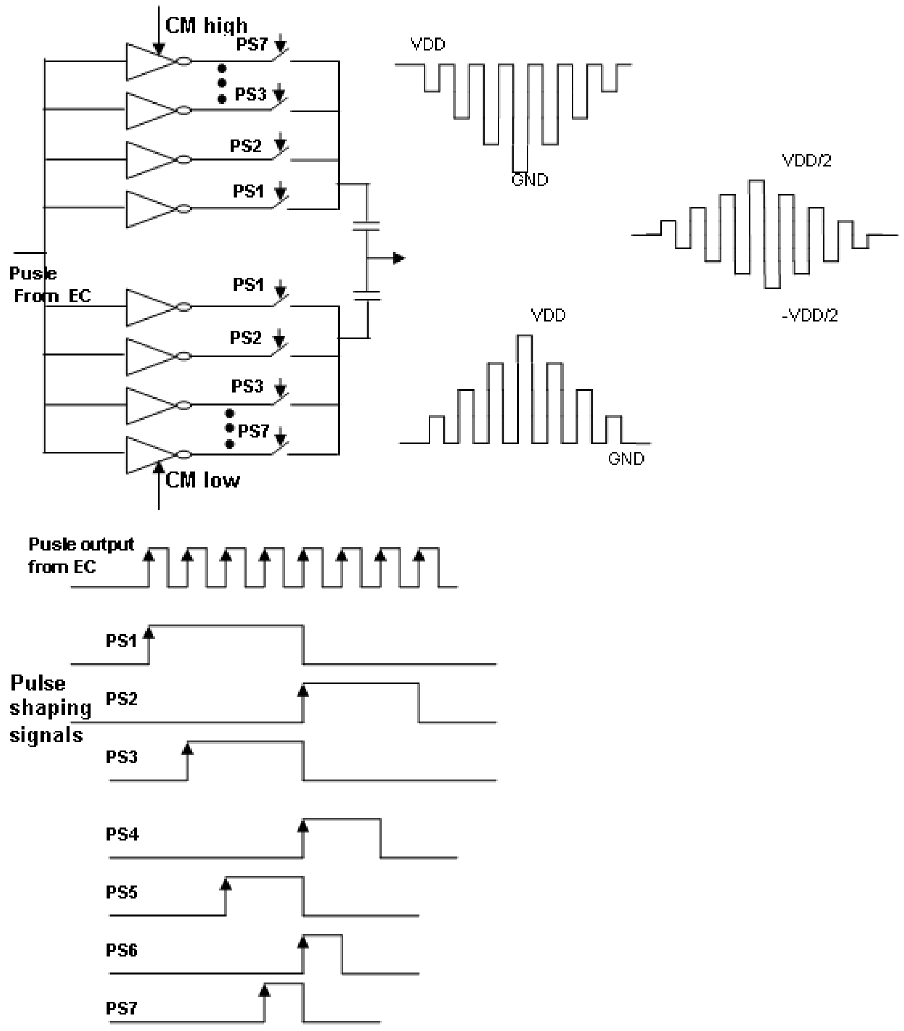


Figure 5.22: Pulse shaping logic

The edge combiner generates the RF pulse signal. Feeding the same RF pulse signal to both top and bottom paths of the digital PA ensures the in-phase RF signal. The differential baseband is generated by ensuring that the outputs of the two PAs are at opposite supply rails immediately before and after pulse generation [7]. The pulse shaping logic will generate the signals PS1 to PS7 as shown in Figure 5.22. These pulse shaping signals along with edge combiner output are fed to the digital PA. Since both capacitors are charging and discharging at the same rate, they approach the same average value during pulse generation. This can be seen as a low frequency transient at the two nodes A and B. However, the changes in average value in both paths are same. If the two paths match, then the low frequency transients get attenuated giving the output UWB pulse at node C with lesser low frequency components [7]. The shape of the pulse can be altered by selecting the number of inverters that would be ON at a particular time through the MUX network [7]. The result obtained with a simple four level shaping of the output pulse from Cadence simulation is as shown in Figure 5.23. The loading of the output antenna was modeled with a 50 ohms resistor in the simulations.

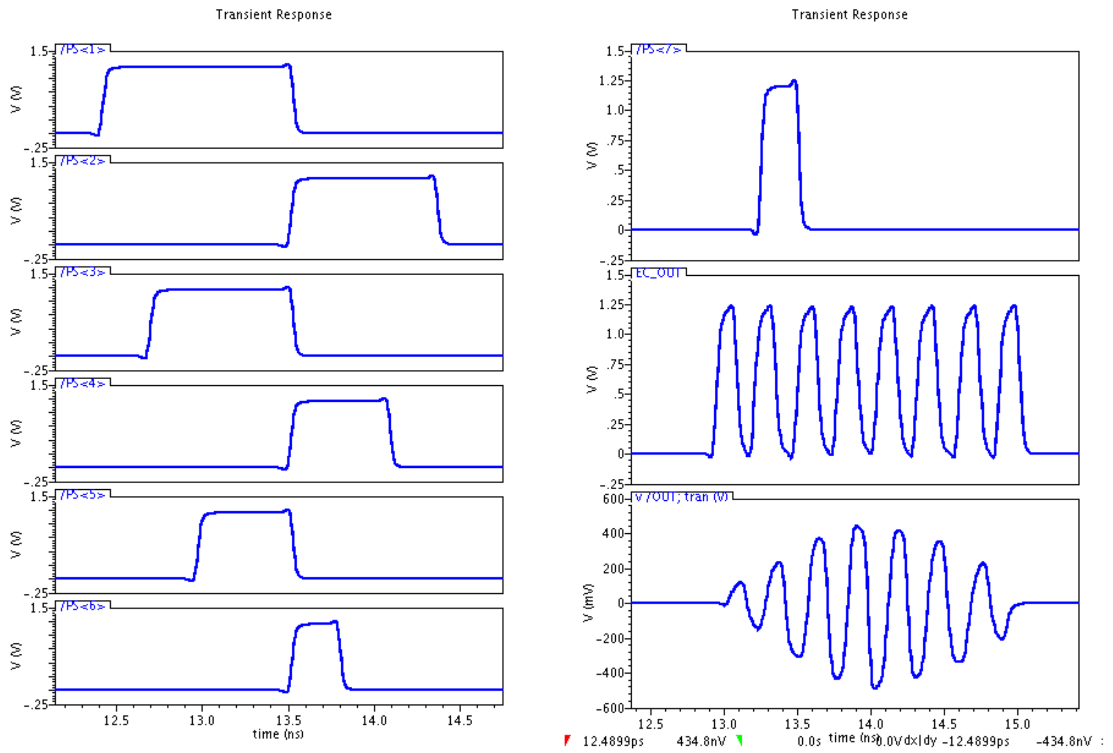


Figure 5.23: UWB four level pulse and pulse shaping signals generated in Cadence

The power spectral density (PSD) plot provided by Cadence will produce the spectrum in terms of dBm/Hz . To obtain the PSD in terms of dBm/MHz, a value of 60dBm needs to be added to the spectrum of the signal. The resulting spectrum of the signal is shown in Figure 5.24. The UWB spectral mask is drawn to show the relative suppression that is required to meet the specifications.

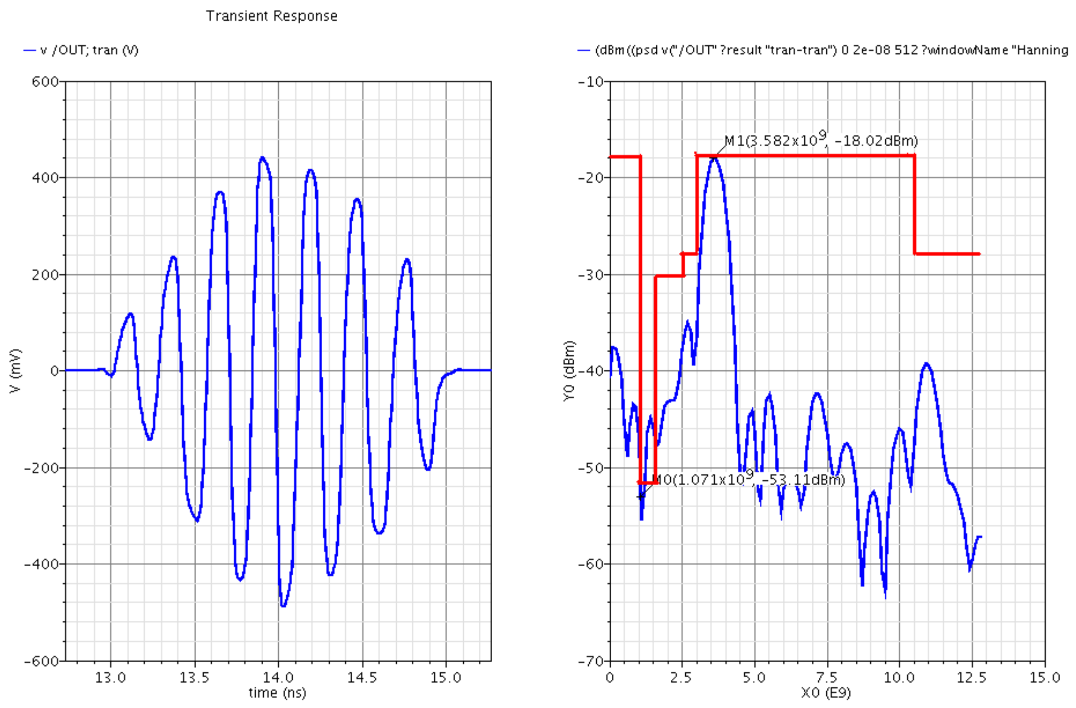


Figure 5.24: Spectrum of the four level UWB pulse generated in Cadence

The spectrum of the output pulse is still not compliant to the FCC mask. The pulse spectrum violates the mask by about 8dB in the GPS band. The system provides the flexibility to control the output pulse shape, pulse width and gain. One such pulse obtained by varying the pulse shape and width is shown in Figure 5.25.

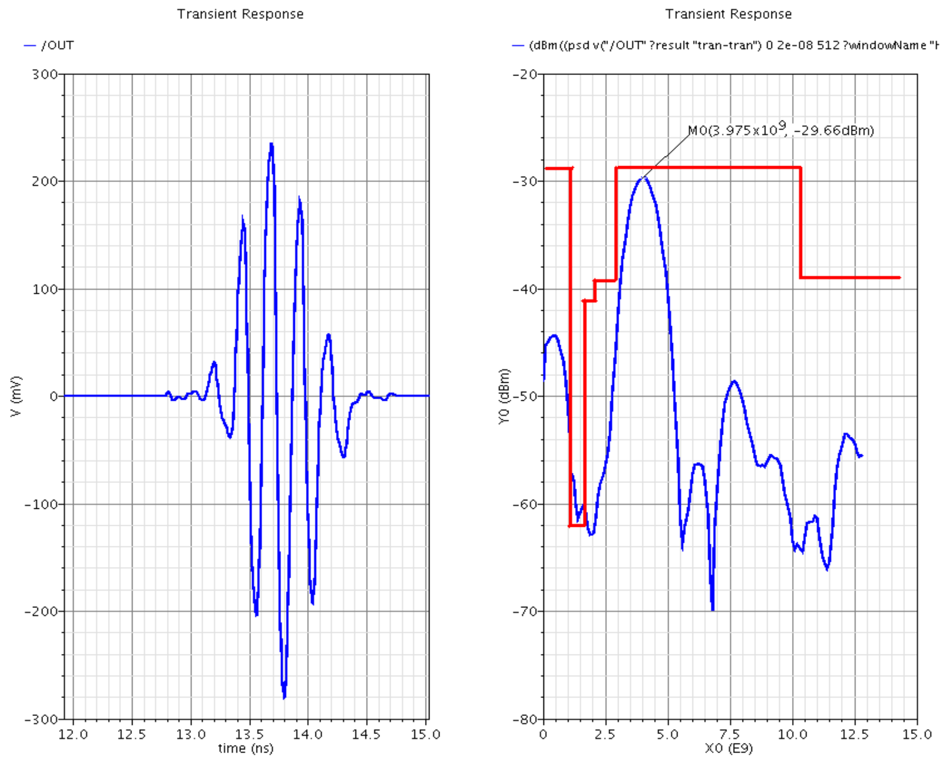


Figure 5.25: Spectrum of the UWB pulse generated in Cadence

The spectrum of this pulse is better than the spectrum of the signal with four level pulse shaping, as it violates the GPS band only by about 4dB. With the level of programmability offered by the system to control the shape, width and gain of the pulse it should be possible to satisfy the UWB mask without using any external filters. The mux network provides the flexibility to digitally control the pulse shape. It provides 2^{28} combinations to tune the pulse shape and make it FCC compliant.

5.6 Full system simulation results and discussion

The complete digital calibration system was simulated using Cadence mixed signal SpectreVerilog simulator. The results obtained from the digital calibration system were given to beamforming UWB transmitter consisting of beam forming delay subsystem and four all-digital UWB transmitters to obtain the required beamforming UWB output. The digital calibration system was programmed for three different configurations, providing a delay difference of 10, 0 and 600ps.

Table 5.3: Inputs used configure the delay difference

Configuration	**Delay difference in ps**	**Centre Frequency In GHZ**	**SEL**	**SDIN**	**Avg. No. of delay cells**	**Exp. delay/cell in ps**
1	10	3.5	4	0	11.5	434.78
			4	-4	11.25	444.44
2	0	4	4	0	11.5	434.78
			4	0	11.5	434.78
3	600	4	9	0	16.5	303.03
			1	-3	8.3125	601.50

In the first configuration, the digital calibration system was programmed to provide an UWB pulse with a centre frequency of 3.5GHz. The beam forming delays were chosen to provide a delay difference of 10ps. The SEL and SDIN inputs were used to set the beam forming delay difference at 10ps. The inputs used and the expected delay values are indicated in Table 5.3. From the table, it can be observed that the expected delay difference between the two delay values is around 10ps. Thus, the beamforming UWB transmitter is expected to provide four UWB signals with a relative delay of 10ps.

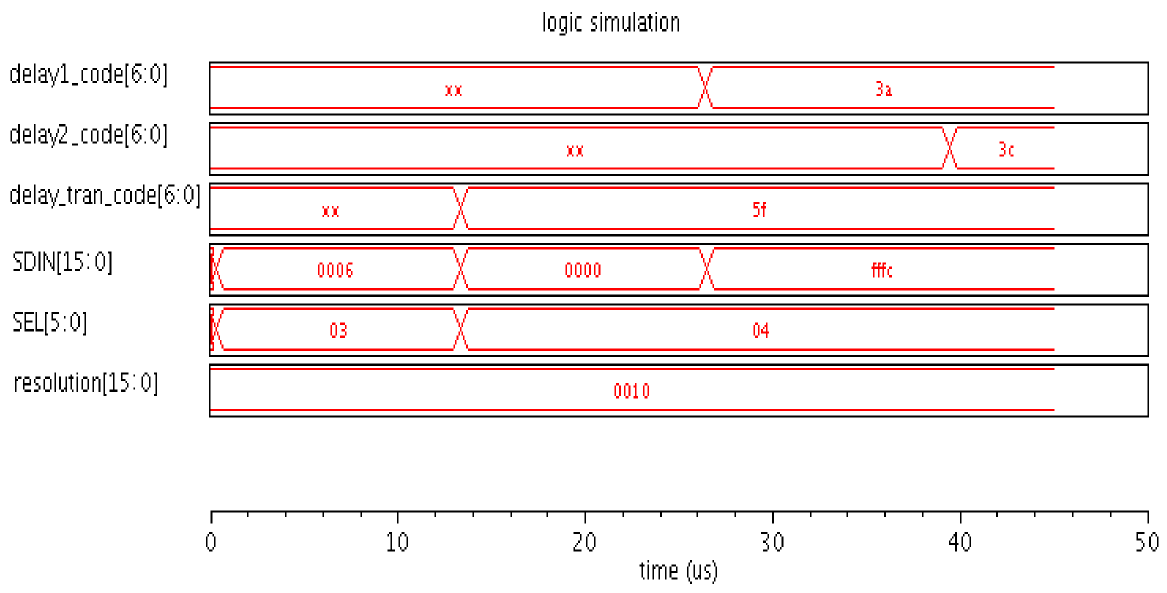


Figure 5.26: Digital calibration result for a configuration to obtain 10ps delay difference

The result of the digital calibration system is shown in Figure 5.26. The calibration system obtained the digital code words to set the transmitter centre frequency and the delay difference of the beamforming delay subsystem. The digital code word (delay_tran_code) to be used to set the centre frequency at 3.5 GHz is obtained as 5F (1011111). The digital code words (delay1_code and delay2_code) for the 2 delay values to be used for the beamforming delay subsystem are obtained as 3A (0111010) and 3C (0111100). Feeding these code words to the beamforming UWB transmitter the resulting output is shown in Figure 5.27.

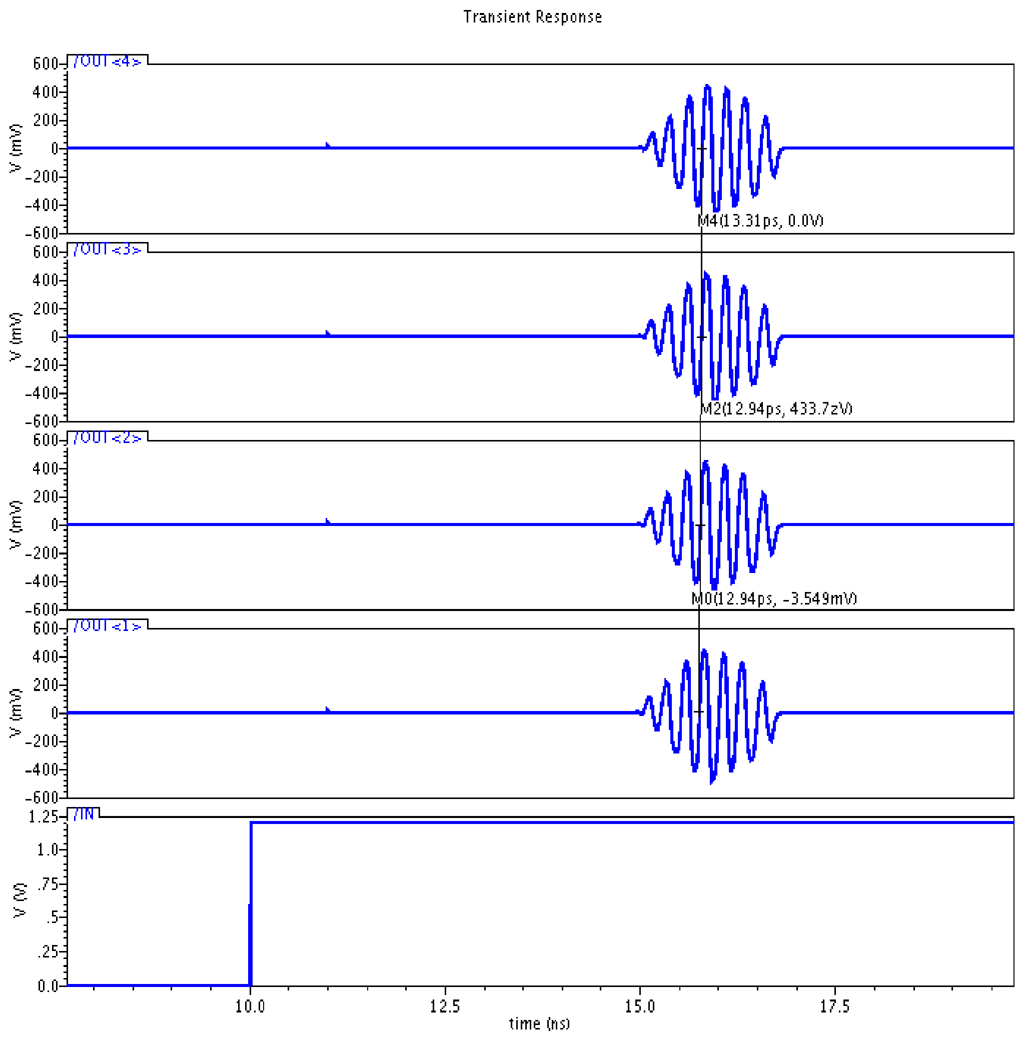


Figure 5.27: Simulated output of the four UWB transmitters showing 13ps delay difference

The simulation result illustrates that the system is able to generate four UWB pulses with a delay difference between them close to 13ps. The delay difference between the signals decides the beam angle. The obtained delay difference value is very close to the theoretically expected delay difference value of 10ps. With an antenna spacing of 18cm, a delay difference of 13ps corresponds to a steering angle of close to 1.3 degrees. Thus, the above simulation clearly depicts the capability of the architecture to obtain delay difference in the range of picoseconds. The centre frequency of the UWB pulse obtained can be verified by looking at the spectrum of the pulse depicted in Figure 5.28.

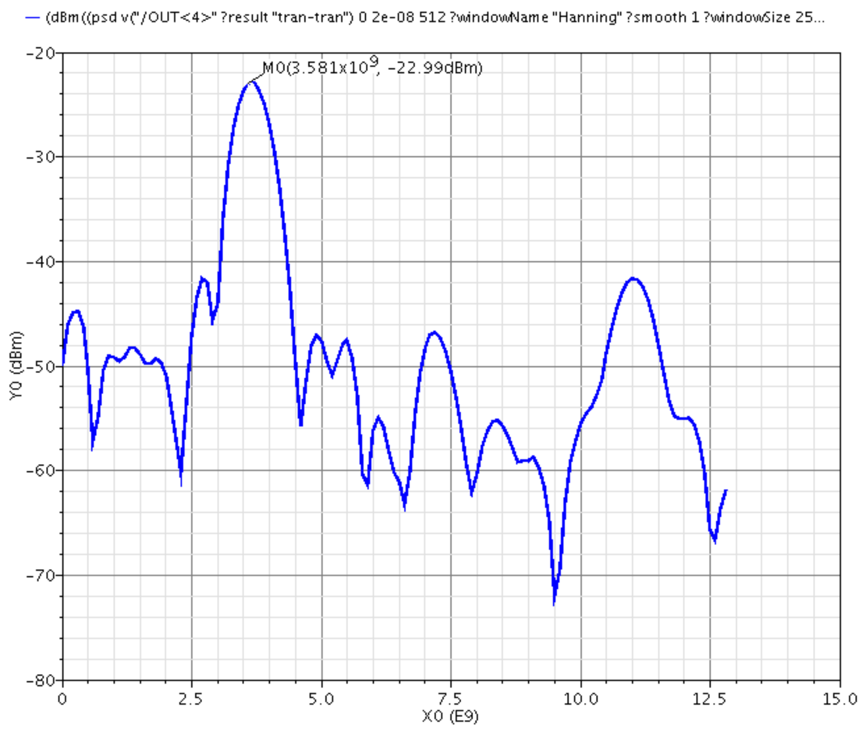


Figure 5.28: Spectrum of UWB pulse showing the centre frequency is 3.581GHz.

The obtained centre frequency of 3.581GHz is quite close to the required centre frequency of 3.5GHz. The centre frequency is off from the desired centre frequency by just 2.3%.

In the second configuration, the system is configured to provide the same digital code word for both the delay elements of the beamforming delay subsystem. Hence, a delay difference very close to 0ps can be obtained. The centre frequency in this case was configured to be 4GHz.

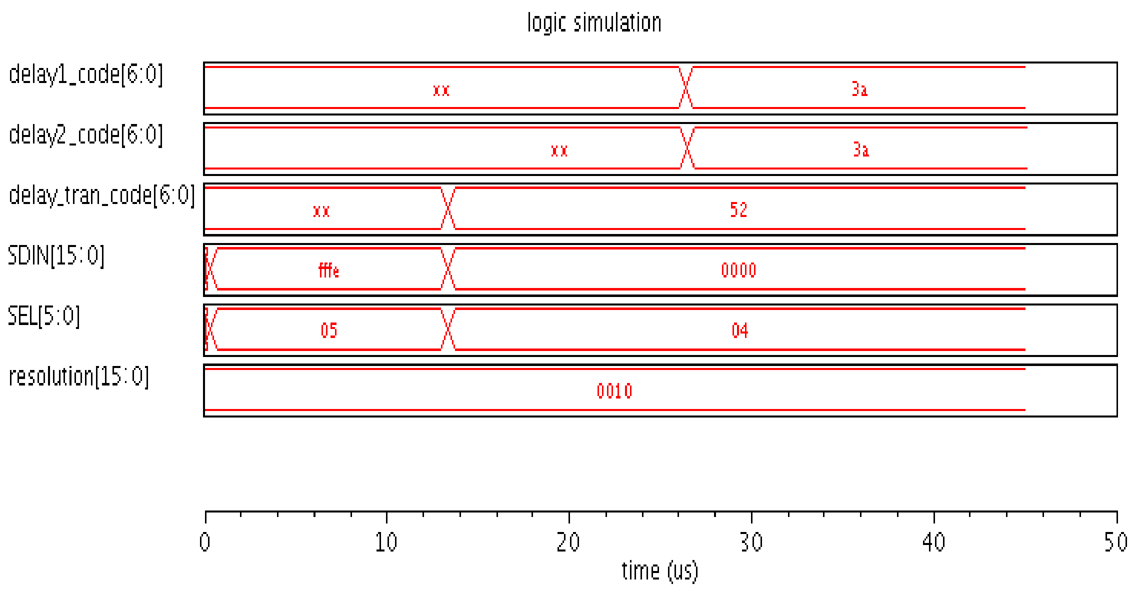


Figure 5.29: Digital calibration result for a configuration to obtain 0ps delay difference

The simulation result obtained from the digital calibration system is shown in Figure 5.29. The result obtained by providing the same digital code word to beamforming delay subsystem of the beamforming UWB transmitter is depicted in Figure 5.30.

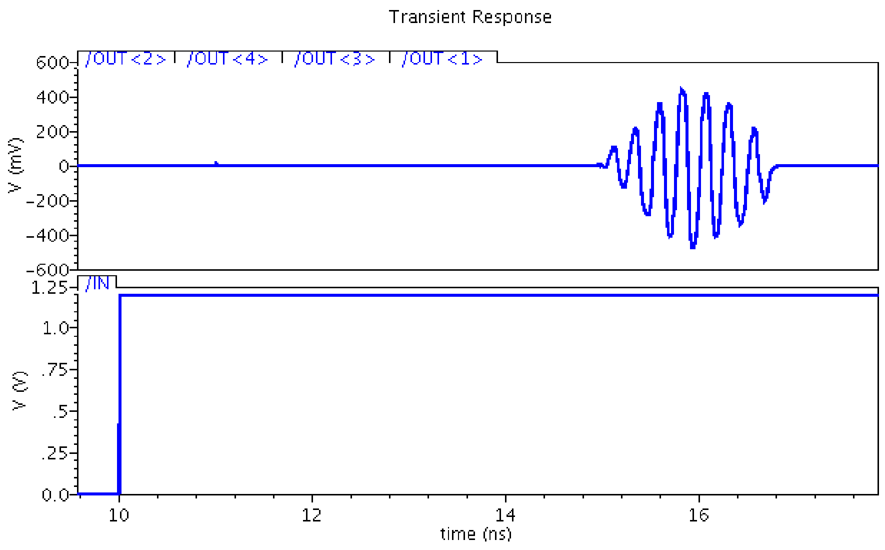


Figure 5.30: Simulated output of the four UWB transmitters showing 0ps delay difference

The output UWB pulses obtained from the four transmitters have been superimposed in Figure 5.30. The perfect overlapping of these signals shows that the delay difference between them is close to 0ps.

In the third configuration, the system was configured to provide a delay difference of 600ps with a centre frequency of 4GHz. From Table 5.3, it can be observed that the inputs used in this case correspond to a delay difference of close to 300ps. By using the range select signal in the beam forming subsystem, the delay range has been doubled to provide the 600ps delay difference. The simulation result of the digital calibration system obtained for this configuration is depicted in Figure 5.31.

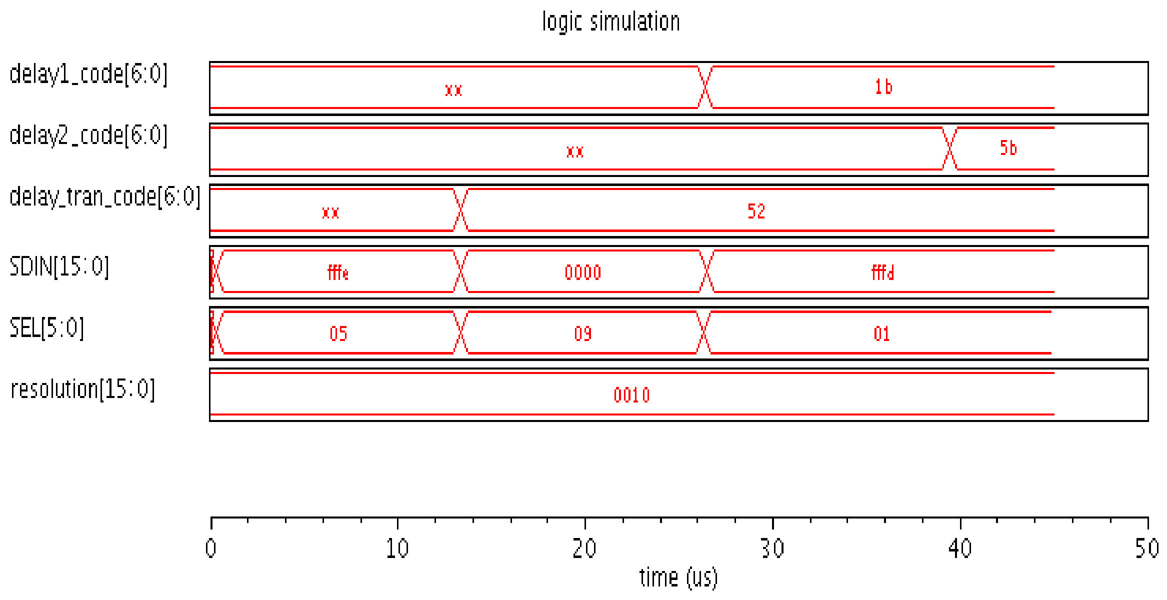


Figure 5.31: Digital calibration result for a configuration to obtain 600ps delay difference

The UWB pulses obtained from the transmitter for this configuration are exhibited in Figure 5.32.

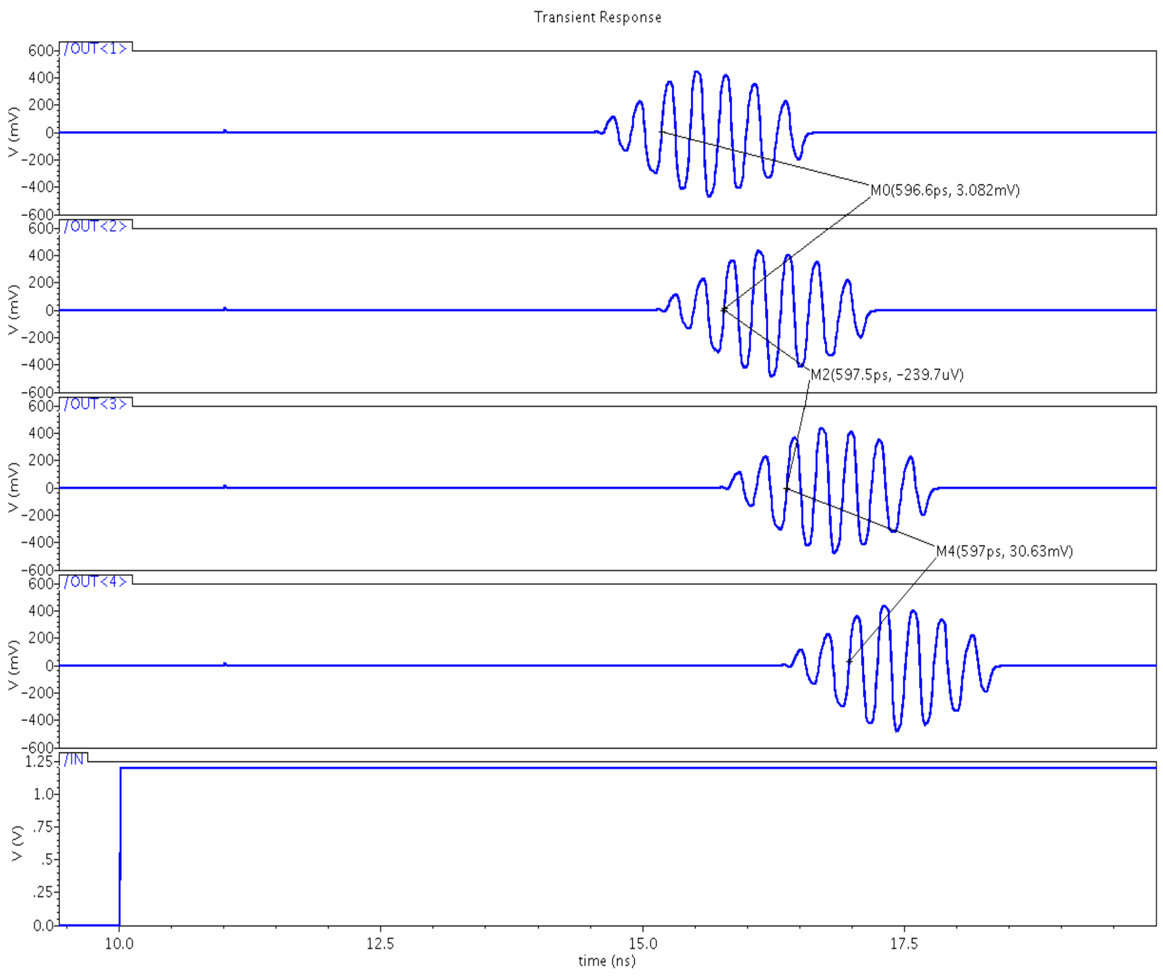


Figure 5.32: Simulated output of the four UWB transmitters showing 597ps delay difference

This result illustrates that the delay difference obtained between the paths is close to 597ps. This value is very close to the expected delay difference of 600ps. The spectrum of the output pulse indicated by Figure 5.33 shows the centre frequency to be 4.074GHz. The centre frequency in this case is off by just 2%.

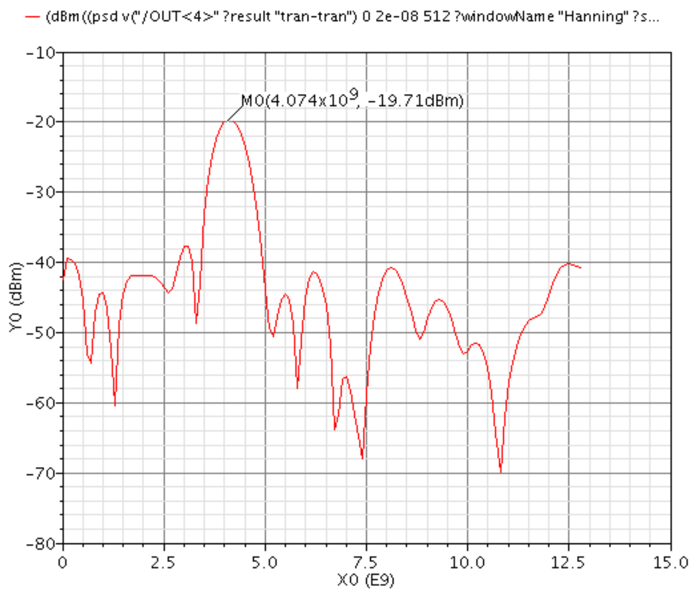


Figure 5.33: Spectrum of UWB pulse showing the centre frequency is 4.074GHz.

Thus, the UWB beamforming transmitter has the capability to generate a delay difference in the range of close to 0ps to 600ps. It can achieve a delay resolution of less than 10ps. It can also operate in three bands with centre frequencies of the 3.5,4 and 4.5GHz and the spectrum of the UWB pulses obtained with these centre frequencies have been superimposed and is depicted in Figure 5.34.

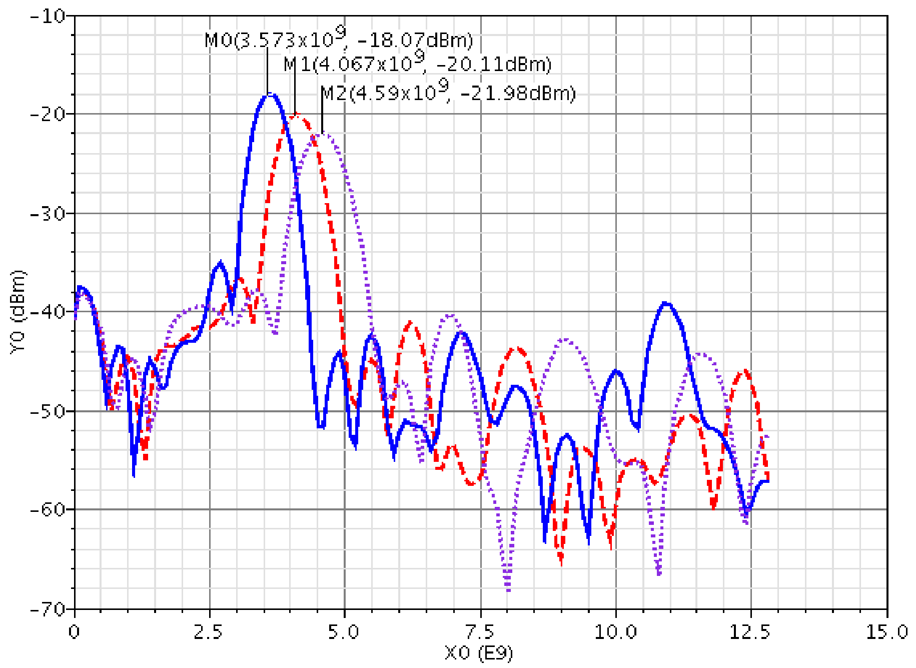


Figure 5.34: Spectrum of the output pulse with three different centre frequencies

Table 5.4: Power consumption of the blocks of all-digital UWB transmitter

BLOCK	**Power Consumption**
Transmitter delay Line	0.50mW
Edge Combiner	1.43mW
Pulse shaping logic	1.08mW
Digital PA with mux network	0.99mW
Total power	4.00mW

The power consumption of the various blocks of a single all-digital UWB transmitter implemented in this design is listed in Table 5.4. The tabulated power consumption is the average power estimated at a data rate of 50MHz assuming a data stream with equal number of ones and zeros with OOK modulation. The output UWB pulse has a voltage swing of 927mV with a pulse width of 1.75ns. Obtaining such a high output voltage is essential to increase the transmission distance. A comparison of the simulated results of the proposed all-digital UWB transmitter with the other existing UWB transmitters is presented in Table 5.5.

Table 5.5: Comparison of a single all-digital UWB transmitter with other UWB transmitters

Ref.	**process**	**Vdd [V]**	**Data Rate [MHz]**	**Modulation**	**Output pulse Voltage Vp-p [mV]**	**Power [mW]**	**E/pulse [pJ]**
[25]	0.18µm	1.8	50	OOK	35	12.6	252
[26]	0.18µm	1.8	400	PPM	210	46	90
[27]	90nm	1.0	1800	BPSK+PPM	220	227	126
[28]	0.18µm	2.2	36	BPSK	640	29.7	825
[35]*	0.13µm	1.2	160	BPSK+PPM	450	10	62.5
[36]*	0.18µm	1.8	1000	OOK	1.8	50	50
[37]	0.18µm	1.5	200	OOK	110	3.4	16.8
[2]	90nm	1.0	10	PPM+DB-BPSK	600	.47	47
[7]	90nm	1.0	15.6^	PPM+BPSK	600	4.4	17.5
This work (simulated)	0.13µm	1.2	50	OOK	927	4	80

*only pulse generator
^transmits a burst of 16 pulses. The value indicated is for a single pulse.

The energy per pulse parameter is defined as the ratio of power consumption to the data rate. From the comparison, it can be seen that the proposed all-digital UWB transmitter used in this work consumes lesser power than most of the other transmitters. It also consumes lesser energy per pulse than some of the other existing transmitters. In [35, 36], the power value mentioned here is only for the pulse generator. Though the architectures [35, 36, 37] consume lesser energy per pulse, they need either an inductor or a balun to form the UWB pulse which might incur large area penalty for beamforming applications. The primary advantage of the all-digital UWB transmitter used in this work is that it avoids the use of inductors and hence would consume less area. This is very crucial for beam forming applications, where multiple instances of the transmitter are required. A comparison with [2, 7] shows that the architectures used in them consume lesser energy/pulse than the transmitter used in the current design. This may be attributed to the better process and lower supply voltage used by these designs.

The beamforming delay subsystem consumes 4.86mW of power and the whole beamforming UWB transmitter consisting of four all-digital UWB transmitters and the beamforming delay subsystem consumes about 35.4mW of power. This value does not include the power consumption of the digital calibration system. The simulation results have already demonstrated the capability of the proposed beamforming UWB transmitter to generate a delay resolution of less than 10ps. A comparison of this beamforming UWB transmitter with the other existing beamforming UWB transmitters is presented in Table 5.6.

Table 5.6: Comparison with other beamforming UWB transmitters

	[4]	[5]*	[1]	[6]	This work(simulated)
Implementation of beamforming	RF	RF	Baseband	Baseband	Baseband
Delay element architecture	Trombone delay element	Path sharing- Constant k-LC Ladder	Phase shifter controlled by PLL	Path sharing- inverter chain	Vernier delay cell calibrated by DLL
Delay resolution	4ps	15ps	100ps	180ps	<10ps
Angular scanning resolution	7°	9°	9°	10°	<1°^
Area	2.25mm^2	9.92mm^2	.9mm^2	-	N/A
Power	87mW	555mW	100mW	-	35.4mW
Antenna spacing	10mm	3cm	18cm	30cm	18cm^

*receiver
^theoretically estimated values

From the comparison, it can be observed that the proposed beamforming UWB transmitter architecture is capable of generating a delay resolution of less than 10ps. This resolution is better than value achieved by existing beamforming transmitters that utilize baseband processing. Thus, with the simulated delay resolution of less than 10ps and by assuming an antenna spacing of 18cm, a theoretical angular resolution of less than 1 degree can be achieved. This value is at least 9 times better than [1]. Also, the simulated power consumption of the transmitter is lower than all of the other transmitters indicated in the table. The delay resolution generated by this transmitter is comparable to the resolution generated by transmitters implementing beamforming at RF. The current design uses an architecture which does not require any inductors. Hence, it would not require a huge area as the architectures described in [4, 5].

CHAPTER 6 : CONCLUSION

6.1 Thesis summary

Ultra-wide band technology is appropriate for a wide variety of applications. The wide bandwidth offered by the technology makes it suitable for high precision imaging and ranging applications. The use of IR-UWB signaling results in energy efficient architecture. These advantages offered by UWB provide more freedom to the designer to build a beamforming transmitter architecture which would occupy less area and consume low power.

This thesis presented the design and implementation of a beamforming UWB transmitter architecture that has the capability to provide very good angular resolution. The system utilized a novel digital delay calibration system to obtain a very fine delay resolution. The delay resolution demonstrated by the transmitter in simulations is of the order of 10ps which gives an angular resolution in the order of 1 degree. This is a significant improvement over the existing beamforming transmitters. The back bone for achieving such a good delay resolution was the capability provided by the sigma-delta DLL, SAR DLL and vernier delay cell to produce a very precise delay. The transmitter operates in the 3-5GHz low band of UWB as specified by the IEEE802.15.4a proposal. The transmitter can operate with centre frequencies of 3.5,4 and 4.5GHz. The non-coherent signaling used in the transmitter relaxes the centre frequency tolerances. The transmitter also provides more flexibility to shape the output pulse to obtain a FCC compliant UWB signal without the need for an external filter. The whole architecture does not require any inductors or external filters to obtain the UWB pulse, thereby resulting in a compact design. The beamforming UWB transmitter consumes a power of about 35.4mW in simulation. This value is lower than most of the existing beamforming UWB transmitters. The single all-digital UWB transmitter used in this design achieves an energy efficiency of 80pJ/pulse when operating at a data rate of 50Mbps.

6.2 Future recommendations

UWB radios are an active area of research and offers potential solution for wireless communications, sensor, biomedical and imaging applications. The beamforming UWB transmitter presented in this thesis offers a solution for high precision imaging applications. Currently, the output UWB pulse generated violates the FCC mask. But this beamforming UWB transmitter has the flexibility to achieve a FCC compliant UWB pulse. Various combinations of pulse shapes can be tried out to obtain a FCC compliant UWB pulse. A Monte-Carlo simulation of the digital power amplifier can give clear picture of the variation of the output spectrum with mismatches. The current transmitter design supports only OOK modulation. If the transmitter supports PPM and BPSK modulations then it could help to improve the spectral characteristics of the output. Although the energy per pulse of the all-digital UWB transmitter is comparable to most of the existing transmitter architectures; there are still rooms for optimization. The edge combiner and the pulse shaping logic blocks used in this design can be optimized to obtain a more energy efficient transmitter. The sigma-delta modulator and the digital calibration subsystem were implemented using verilog. The digital synthesis of these blocks would give a better estimate of the angular resolution capability of the beamforming transmitter.

The mask layout of the entire UWB beamforming transmitter still needs to be done to fully verify the concept through post layout simulation and actual chip testing.

References

[1] M.Y.W. Chia, T. H. Lim, J.K. Yin, P.Y. Chee,S.W. Leong and C.K. Sim, "Electronic Beam Steering Design for UWB Phased Array", *IEEE Trans.Microwave, Theory and Tech.*, vol. 54, Jun. 2006, pp. 2431-2438.

[2] D. D. Wentzloff and A. P. Chandrakasan, "A 47 pJ/pulse 3.1–5 GHz all-digital UWB transmitter in 90 nm CMOS", *IEEE ISSCC Dig. Tech. Papers*, 2007, pp. 118–119.

[3] P. P. Mercier, D. C. Daly, and A. P. Chandrakasan, "A 19 pJ/pulse UWB transmitter with dual capacitively-coupled digital power amplifiers", *IEEE RadioFrequency Integr. Circuits Symp.*, 2008, pp. 47–50.

[4] J. Roderick, H. Krishnaswamy, K. Newton, and H. Hashemi, "Silicon-Based ultra-wideband beam-forming", *IEEE Journal of Solid-State Circuits*, vol. 41, no. 8, Aug. 2006, pp. 1726–1739.

[5] T. Chu, J. Roderick, and H. Hashemi, "An integrated ultrawideband timed array in 0.13um CMOS using a path sharing true time delay architecture," *IEEE Journal of Solid-StateCircuits*, vol. 42, no. 12, Dec. 2007, pp. 2834-2850.

[6] Z. Safarian,T.S. Chu and H. Hashemi "A 0.13μm CMOS 4-channel UWB timed array transmitter chipset with sub-200ps switches and all-digital timing circuitry", *IEEE RFIC Symp. Dig.*, 2008, pp. 601-604

[7] P. Mercier et al. "Energy-efficient All-Digital UWB Transmitter employing Dual capacitively coupled pulse shaping drivers", *IEEE Journal of Solid- State Circuits*, vol. 44, no. 6, Jun. 2009, pp.1679-1688.

[8] [Online]. Available: http://www.lifewaveinc.com/

[9] FCC, Revision of part 15 of the commission's rules regarding ultra-wideband transmission systems, FCC 02-48, Feb. 2002.

[10] A. Hajimiri, H. Hashemi, A. Natarajan, X. Guan, Komijani, "Integrated phased array systems in silicon", *Proceedings of the IEEE*, vol. 93, no. 9, Sep. 2005 pp.1637 – 1655

[11] J. Paramesh, R. Bishop, K. Soumyanath, and D. Allstot, "A 1.4 V 5GHz four-antenna Cartesian combining receiver in 90 nm CMOS for beamforming and spatial diversity applications", *IEEE ISSCC Dig.Tech. Papers*, Feb. 2005, pp. 210–212.

[12] David. D. Wentzloff, "Pulse–based ultra wideband transmitters for digital communication", Ph.D Thesis, MIT, Jun. 2007.

[13] M. G. M. Hussain, "Antenna patterns of non-sinusoidal waves with time variation of Gaussian pulse", *IEEE Transactions on Electromagnetic Compatibility,* vol 30, no. 4, Feb. 1989, pp. 504-512

[14] S. Werner, C. Strum, and W. Wiesbeck "Impulse response of linear UWB antenna arrays and the application to beam steering", *Proc. Int. Ultrawideband Conf.*, Sep. 2005, pp. 275–280.

[15] M. G. M. Hussain, "Principles of space-time array processing for ultrawide-band impulse radar and radio communications," *IEEE Transactions. Veh. Technol.*, vol. 51, May 2002, pp. 393–403.

[16] T. Nilsson, C. Samuelsson, M. Alfredson, and A. Quacha, "Key circuits for a reconfigurable and bi-directional beam former for ultra wideband applications", *Eur. Microwave Conf.*, Munich, Germany, Oct. 2003, pp. 735–738.

[17] Xiang Guan, H. Hashemi, A. Hajimiri, "A Fully Integrated 24-GHz Eight-Element Phased-Array Receiver in Silicon", *IEEE Journal of Solid-State Circuits,* vol. 39, no. 12,Dec. 2004, pp. 2311 - 2320

[18] B. Miller, B. Conley, "A multiple modulator fractional divider", *Annual IEEE Symposium on Frequency Control,* vol. 44, Jun. 1991, pp. 578-583.

[19] T. A. Riley, M. A. Copeland and T. Kwasniewski, "Delta-sigma modulation in fractional-N frequency synthesis", *IEEE Journal of Solid-State Circuits,* vol. 28, no. 5, May 1993, pp. 553-559.

[20] R. Jacob Baker, "*CMOS circuit design, layout and simulation*",2nd Edition, IEEE press series on microelectronic systems, 2005, pp. 592-602.

[21] Yang Rong-Jyi, Liu Shen-Iuan "A 2.5 GHz All-Digital Delay-Locked Loop in 0.13 µm CMOS Technology**"**, *IEEE Journal of Solid-State Circuits,* vol. 42, no. 11, Nov. 2007, pp. 2338 – 2347

[22] Yang, R.-J.; Liu, S.-I. "A 40–550 MHz Harmonic-Free All-Digital Delay-Locked Loop Using a Variable SAR Algorithm**"**, *IEEE Journal of Solid-State Circuits,* vol. 42, no. 2, Feb. 2007, pp. 361 – 373

[23] Moyer, G.C.; Clements, M.; Liu, W. "Precise delay generation using the Vernier technique", *Electronics Letters,* vol. 32, Issue 18, pp.1658-59

[24] P. Dudek, S. Szczepanski, J.V. Hatfield, "A high-resolution CMOS time-to-digital converter utilizing a Vernier delay line", *IEEE Journal of Solid-State Circuits,* vol. 35, no. 2, Feb. 2000, pp. 240 - 247

[25] Y.Zheng, Y.Tong, Jyan, Y.-P.Xu, W.G.Yeoh, and F.Lin "A low power noncoherent CMOS UWB transceiver ICs", *IEEE RadioFrequency Integr. Circuits Symp,*, Jun. 2005, pp. 347-350

[26] Y. Zheng, Y. Tong, C. W. Ang, Y.-P. Xu, W. G. Yeoh, F. Lin, and R. Singh, "A CMOS carrier-less UWB transceiver for WPAN applications", *IEEE ISSCC Dig. Tech. Papers*, 2006, pp. 378–387.

[27] M. Demirkan and R. R. Spencer, "A pulse-based Ultra-Wideband transmitter in 90-nm CMOS for WPANs", *IEEE Journal of Solid-State Circuits*, vol. 43, no.12, Dec. 2008, pp. 2820–2828.

[28] T. Norimatsu, R. Fujiwara, M. Kokubo, M. Miyazaki, A. Maeki, Y. Ogata, S. Kobayashi, N. Koshizuka, and K. Sakamura, "A UWB-IR transmitter with digitally controlled pulse generator, *IEEE Journal of Solid-State Circuits*, vol. 42, no. 6, Jun. 2007, pp. 1300–1309.

[29] H. H. Chang, et al., " A Wide-Range Delay-Locked Loop With a Fixed Latency of One Clock Cycle", *IEEE Journal of Solid-State Circuits*, vol. 37, no. 8, Aug. 2002, pp.1021-1028,

[30] John G. Maneatis, "Low-jitter process-independent DLL and PLL based on self-biased techniques", *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, Nov. 1996, pp.1723-1725,

[31] Maymandi-Nejad, M. Sachdev, **"**A monotonic digitally controlled delay element**"** , *IEEE Journal of Solid-State Circuits*, vol. 40, no. 11, Nov. 2005, pp. 2212 – 2219

[32] Jabeom Koo, Sunghwa Ok, Chulwoo Kim , "Low-Power Programmable DLL-Based Clock Generator With Wide-Range Antiharmonic Lock**",** *IEEE Transactions on circuits and system II*, vol. 56, no. 1, Jan. 2009, pp. 21-25.

[33] Online " techon.nikkeibp.co.jp/NEA/archive/200205/183143/"

[34] Chulwoo Kim, In-Chul Hwang, Sung-Mo Kang ," A low-power small-area ±7.28-ps-jitter 1-GHz DLL-based clock generator", *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, Nov. 2002, pp. 1414 – 1420

[35] L. Smaini, C. Tinella, D. Helal, C. Stoecklin, L. Chabert, C. Devaucelle, R. Cattenoz, N. Rinaldi, D. Belot, "Single-Chip CMOS Pulse generator for UWB systems", *IEEE Journal of Solid-State Circuits*, vol. 41, no. 7, Jul. 2006, pp. 1551 – 1561

[36] Y. Zhu, J. D. Zuegel, J. R. Marciante, and H. Wu, "A 10 GS/s distributedwaveform generator for sub-nanosecond pulse generation andmodulation in 0.18 um standard digital CMOS," *IEEE RadioFrequency Integr. Circuits Symp.*, Jun. 2007, pp. 35–38.

[37] Tuan-Anh Phan, Krizhanovskii, Sang-Gug Lee "Low-Power CMOS Energy Detection Transceiver for UWB Impulse Radio System", *IEEE Custom Integr. Circuits Conf.*, Sep2007, pp.675-678.

APPENDIX A: MATLAB MODELS AND CODES

The entire beamforming UWB transmitter was modeled using MATLAB-SIMULINK. The model was split into two: namely, digital calibration subsystem model and the beamforming UWB transmitter model. These models developed in SIMULINK, are depicted in the following figures. The digital calibration subsystem model is shown in Figure A.1

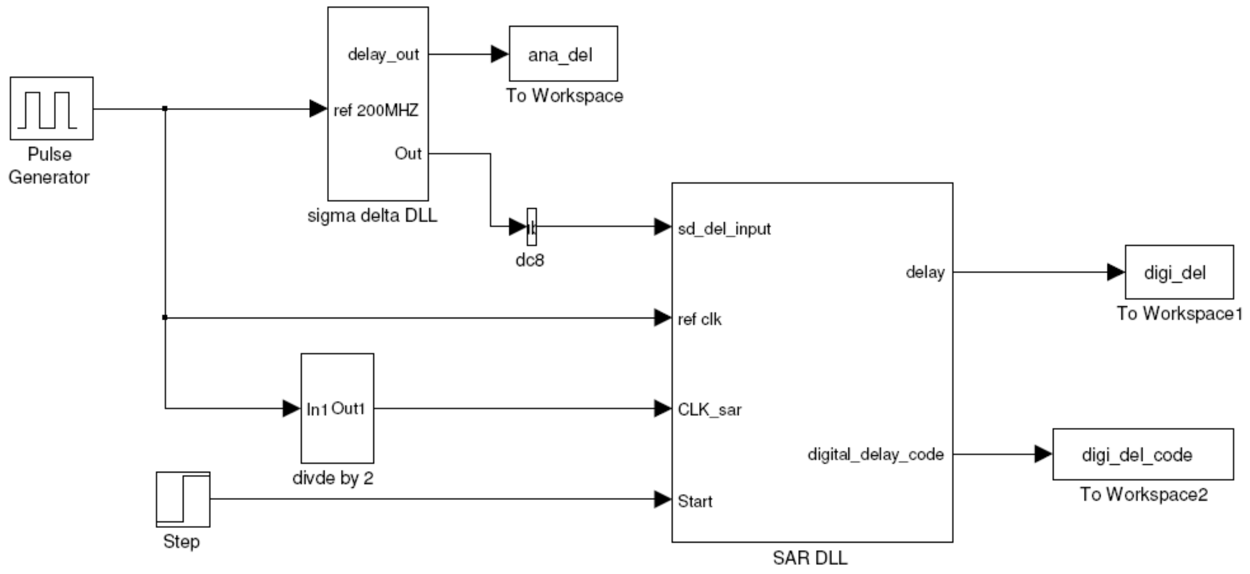


Figure A.1: Digital calibration subsystem -SIMULINK model

This consists of a sigma-delta DLL and a SAR DLL. It basically takes the input to the sigma-delta DLL and produces the required analog delay. The SAR DLL then obtains the digital code word that would produce this delay. The sigma-delta DLL model is illustrated in Figure A.2.

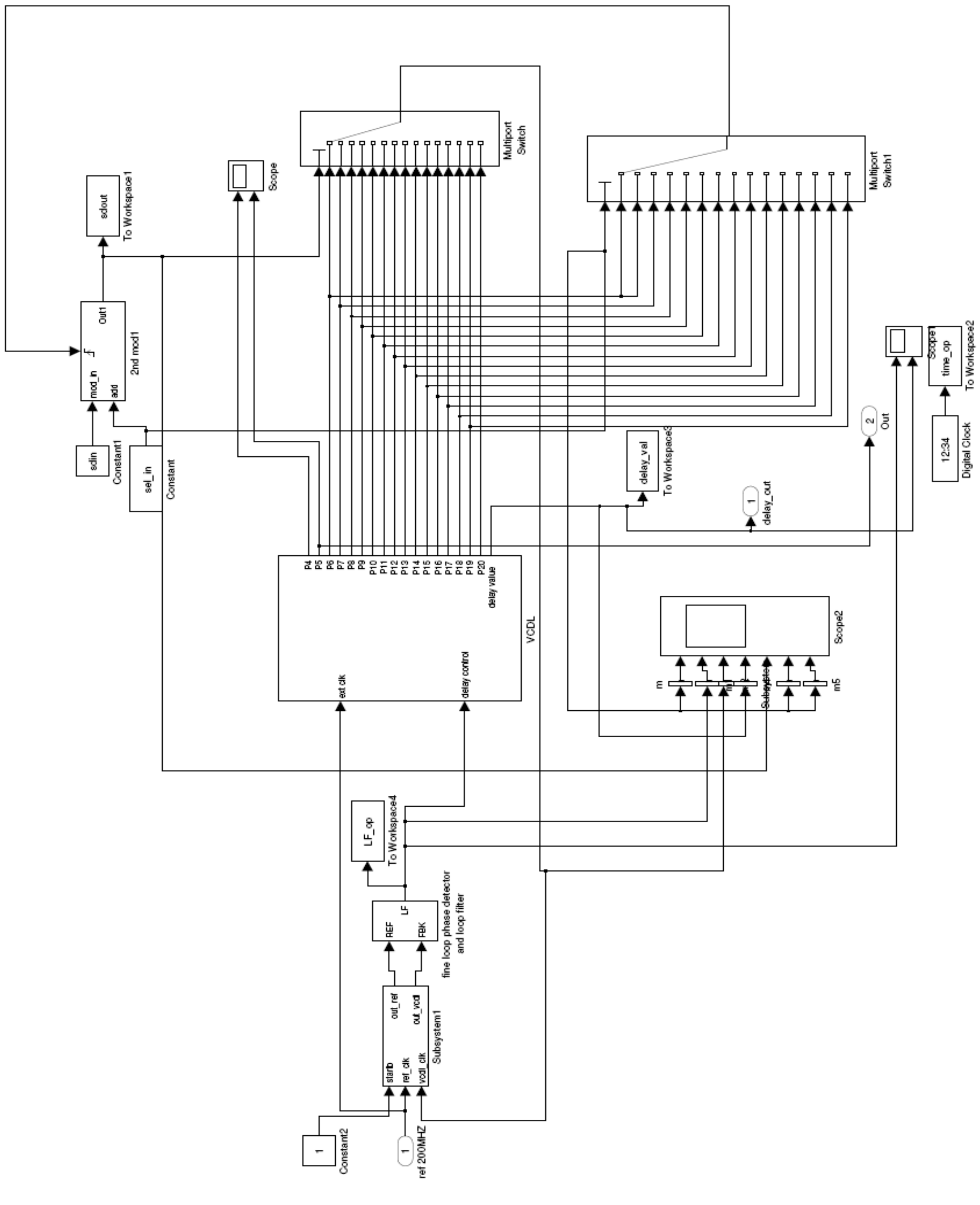


Figure A.2: Sigma-delta DLL-SIMULINK model

The sigma-delta DLL generates the required analog delay. It takes two inputs namely, ‘sdin’ and ‘sel_in’ that decide the analog delay to be generated. The SAR DLL shown in Figure A.3 obtains the digital code word that needs to be used to obtain a delay close to the delay generated by the sigma-delta DLL. The SAR DLL converts the digital code word into a decimal value and provides it to the output.

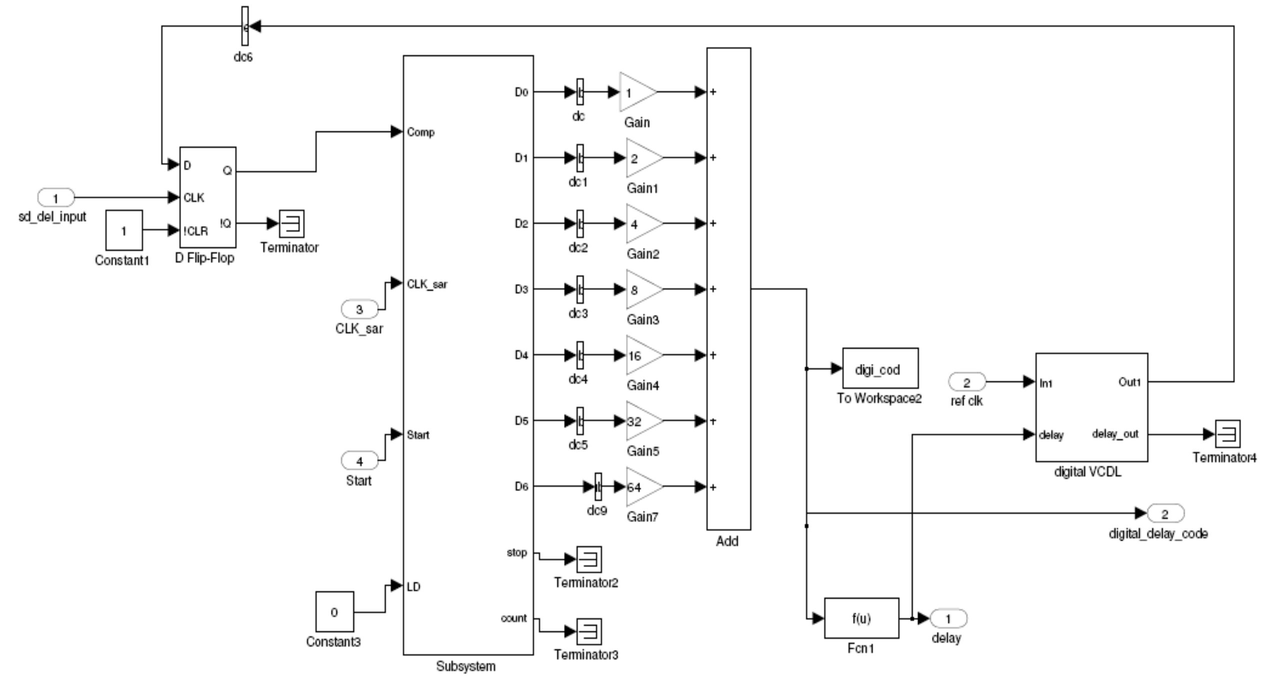


Figure A.3: SAR DLL –SIMULINK model

The inputs corresponding to various delay values are provided with help of a script and the corresponding delay generated by the sigma-delta DLL, the output digital delay value and the digital code word generated by the SAR DLL are recorded by the script. These values are then used in the beamforming UWB transmitter model. The code listing of the MATLAB script is as follows.

digital_calibration_script.m

```
clear;
add_in_array= [enter inputs corresponding to various delays];
sdin_array=[enter inputs corresponding to various delays];
   for index= 1:1:length(adin_array)
         index
         add_in=add_in_array(1,index);
         sdin=sdin_array(1,index);
         tic
         sim('digital_calibration'); % Starts Simulink simulation
         toc
         len_a=length(ana_del);
         len_d=length(digi_del);
         len_c=length(digi_cod);
         div(index,1)=6+mean(sdout1);
         digital_code(index,1)=digi_cod(len_c);
         analog_delay(index,1)=ana_del(len_a)*1e12;
         digital_delay(index,1)=digi_del(len_d)*1e12;
         jit(index,1)=analog_delay(index,1)-
digital_delay(index,1);
end
```

The digital code words obtained from the digital calibration subsystem are used as inputs to the beamforming UWB transmitter model depicted in Figure A.4.

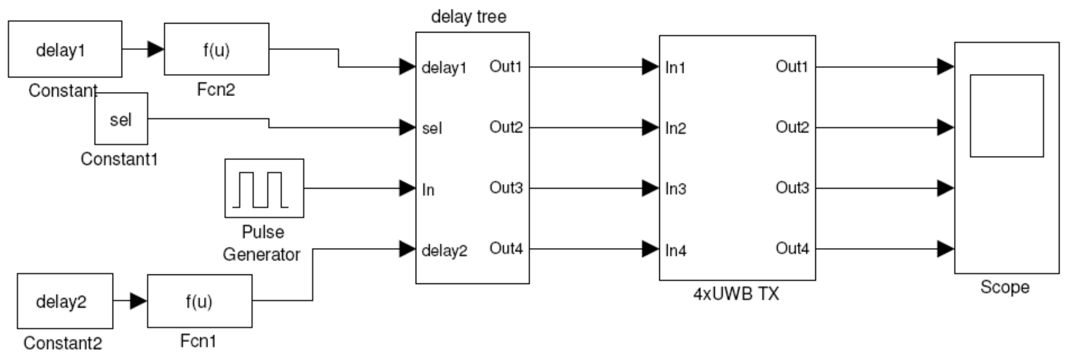


Figure A.4: Beamforming UWB transmitter –SIMULINK model

The model has a delay tree and four UWB transmitters to form the UWB pulses with a four level pulse shaping. The delay tree model is shown in Figure A.5.

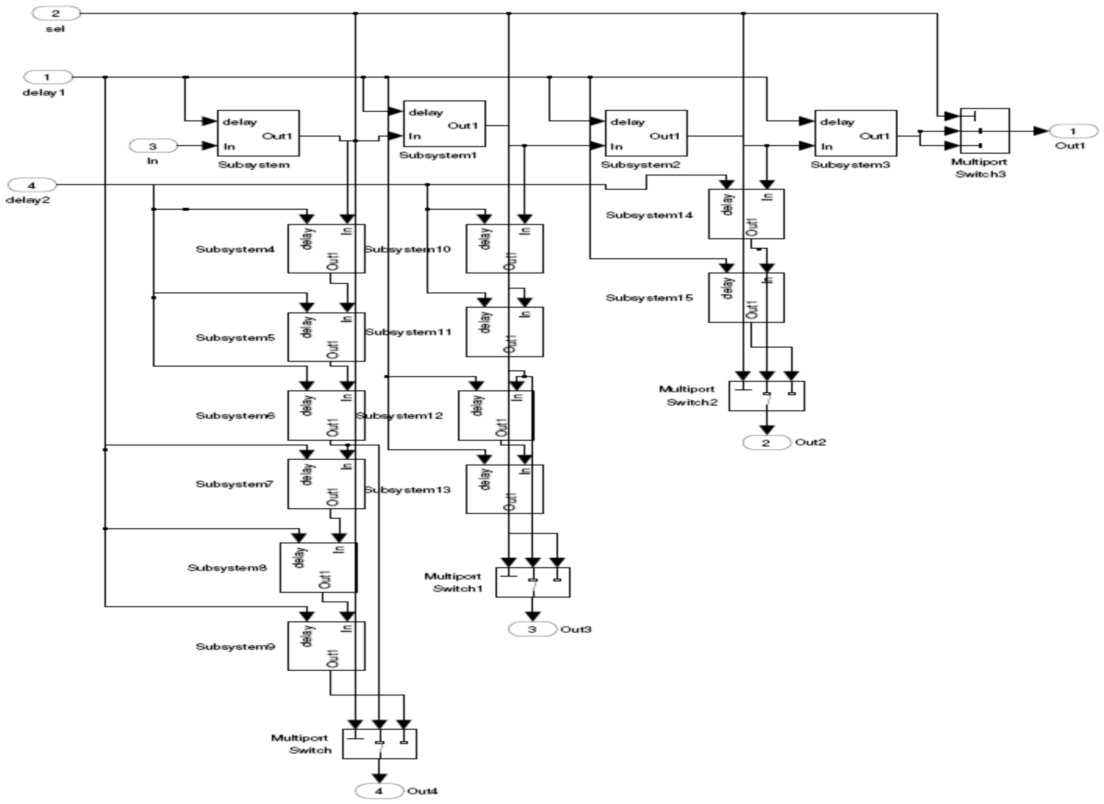


Figure A.5: Delay tree-SIMULINK model

The delay tree basically takes in the digital code words corresponding to the required delay difference. It has a ‘sel’ input that is used to double the delay difference range when needed. It generates the desired delay difference between the four output paths. The four UWB transmitters are modeled as indicated by Figure A.6.

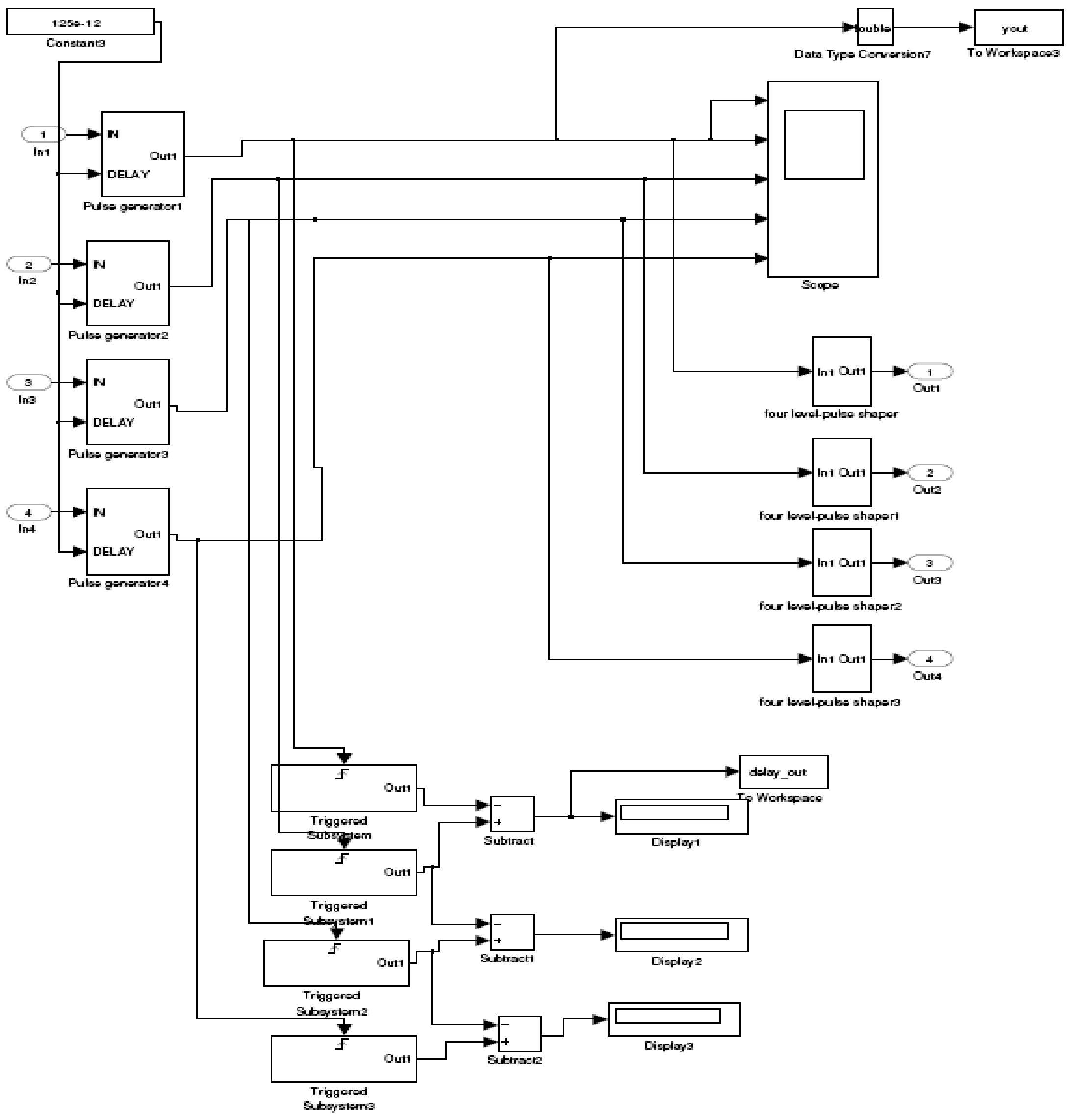


Figure A.6: 4xUWB transmitter-SIMULINK model

Each transmitter has a pulse generator that generates an output pulse with 4GHz frequency. The pulse width can be also be controlled by the transmitter. The output pulses are fed to the four level pulse shaper block that generates the output UWB pulse with four amplitude levels. The inputs to the beamforming UWB transmitter are provided with help of a script. The script sets the delay difference of the model to various values and determines the beam angle. It plots the corresponding antenna pattern. It also obtains the spectrum of the output UWB pulse. The code listing of the script is given below.

Beamforming_UWB_transmitter_script.m

```
ne=4;%number of antenna elements
delay_code=[enter delay codes corresponding to various delays];
 delay1=delay_code(1,1);
 for i=1:1:1
 sel=0;
  if(i>60)
      sel=1;
      delay2=delay_code(1,i-60);
  else
    delay2=delay_code(1,i);
 end
tic;
sim('uwb_transmitter_sys_erf'); % Starts Simulink simulation
toc;
delay=max(delay_out)
%----------------------------------------------------------------
 %PLOT Antenna pattern at the estimated angle
 %---------------------------------------------------------------
beta0=asin(3*1e8*delay/0.18)*180/pi
angles_arr(i,1)=beta0;
eps=0.0000001;
beta=0:pi/10791:2.*pi;
beta0= beta0*pi/180.;
var=sin(beta)-sin(beta0);
T=471e-12; %Pulse width of the signal
c=3e8;      %velocity of light
L=54e-2;    %antenna length with 18cm spacing
num=erf(1.77*L.*var*.5/(T*c));
if(abs(num)<=eps)
    num=eps;
end
den=(1.77*L.*var*.5/(T*c));
if(abs(den)<=eps)
    den=eps;
end
pattern=num./den;
maxval=max(abs(pattern));
pattern=abs(pattern./maxval);
emod=abs(pattern);
Figure(1)
polar(beta,abs(pattern))
end



%------------------------------------------------------------------
 %PLOT THE SPECTRUM OF THE SIGNAL
 %------------------------------------------------------------------
```

```
Fs = 1/1e-12; %sampling rate
Ts = 1/Fs %sampling time interval
Tstop=10e-9;
t = 0:Ts:Tstop; %sampling period
n = length(t); %number of samples
L=length(out);
for i=1:1:L
 z(i,1)=out(1,1,i);
end
y= z;
[YfreqDomain,frequencyRange] = positiveFFT(y,Fs);
Figure(2);
plot(frequencyRange,20*log10(abs(YfreqDomain)));
xlabel('Freq (Hz)')
ylabel('magnitude in Db')

title('spectrum using FFT')
```

APPENDIX B: VERILOG CODES

The second order sigma-delta modulator and the digital calibration FSM with the serial to parallel interface (SPI) were built using verilog and the code listing of these blocks are given below.

Sigma–delta modulator verilog code

```
//Verilog HDL for "sig_del_mod2", "del_sig_mod2" "verilog"

`timescale 1ns/10ps

//4-bit adder
module cla4 (a,b,cin,sum,cout,P,G);
input [3:0] a,b;//input
input cin;    //input carry
output P;   //group propagate
output G;   //group generate
output[3:0] sum;
output cout;
wire P;
wire G;
wire P_0,P_1,P_2,P_3;
wire G_0,G_1,G_2,G_3;
wire c0,c1,c2,c3;
assign P_0 = a[0]^b[0];
assign P_1 = a[1]^b[1];
assign P_2 = a[2]^b[2];
assign P_3 = a[3]^b[3];
assign G_0 = a[0]&b[0];
assign G_1 = a[1]&b[1];
assign G_2 = a[2]&b[2];
assign G_3 = a[3]&b[3];
assign P = P_0 & P_1 & P_2 & P_3;
assign G = G_0 & P_1 & P_2 & P_3 |G_1 & P_2 & P_3 | G_2 & P_3 | G_3;
assign c0 = G_0 | P_0 & cin;
assign c1= G_1 | P_1 & G_0 | P_0 & P_1 & cin;
assign c2 = G_2 | G_1 & P_2 | G_0 & P_1 & P_2 | cin & P_0 & P_1 &
P_2;
assign c3 = G_3 | G_2 & P_3 | P_3 & P_2 & G_1 | P_3 & P_2 & P_1 &
G_0 | P_3 & P_2 & P_1 & P_0 & cin;
assign sum[0] = P_0 ^ cin;
assign sum[1] = P_1 ^ c0;
assign sum[2] = P_2 ^ c1;
assign sum[3] = P_3 ^ c2;
assign cout = c3;
endmodule


//////////////////////////////////////////////////////////////
//             16 bit carry look ahead adder                //
//////////////////////////////////////////////////////////////

module cla16 (a,b,cin,sum,cout16);
```

```
input [15:0] a,b; //input
input cin;        //input carry
output[15:0] sum; //sum
output cout16;      //16 bit output carry
wire[15:0] sum;
wire ca1,ca2,ca3,ca4;


cla4 CLA4_1(.sum(sum[3:0]),.cout(cout0),.P(P_0),.G(G_0),.a(a[3:0]),.b(b[3:0]),.cin(cin));
cla4 CLA4_2(.sum(sum[7:4]),.cout(cout1),.P(P_1),.G(G_1),.a(a[7:4]),.b(b[7:4]),.cin(ca1));
cla4 CLA4_3(.sum(sum[11:8]),.cout(cout3),.P(P_2),.G(G_2),.a(a[11:8]),.b(b[11:8]),.cin(ca2));
cla4 CLA4_4(.sum(sum[15:12]),.cout(cout4),.P(P_3),.G(G_3),.a(a[15:12]),.b(b[15:12]),.cin(ca3));

assign ca1 = cin & P_0 |G_0;
assign ca2 = cin & P_0 & P_1 | G_0 & P_1 | G_1;
assign ca3 = G_2 | G_1 & P_2 | G_0 & P_1 & P_2 | cin & P_0 & P_1 & P_2;
assign ca4 = G_3 | P_3 & G_2| P_3 & P_2 & G_1| P_3 & P_2 & P_1 & G_0 | P_0 & P_1 & P_2 & P_3 & cin;
assign cout16=ca4;
endmodule


////////////////////////////////////////////////////////////////
//               z/(z-1) accumulator                       //
//               1/(1-(1/z))                            //
////////////////////////////////////////////////////////////////
module accumulator1 (x, clk, rstb, y);
input   [15:0] x;//input
input          clk, rstb;
output  [15:0] y; //output

wire    [15:0] y;
wire    [15:0] sum;
wire           nc;
reg     [15:0] yd;   //output after 1 unit delay

always @(posedge clk or negedge rstb)
if (!rstb)
  yd <= 16'b0000000000000000;
else
  yd <= y;

cla16 CLA16_1 (x, yd, 1'b0, y, nc);

endmodule

////////////////////////////////////////////////////////////////
//               1/(z-1) accumulator                       //
////////////////////////////////////////////////////////////////
module accumulator2 (x, clk, rstb, y);
input   [15:0] x;
input          clk, rstb;
output  [15:0] y;

reg     [15:0] y;
wire    [15:0] sum;
buf     (tie0, 1'b0);
wire           nc;

always @(posedge clk or negedge rstb)
if (!rstb)
  y <= 16'b0000000000000000;
else
  y <= sum;
```

```
cla16 CLA16_1 (x, y, 1'b0, sum,nc);
endmodule

//////////////////////////////////////////////////////////////
//          16bit Quantizer                              //
//////////////////////////////////////////////////////////////

module quantizer(quant_in,sel_in,resolution,rstb,quant_out,fb);
input [15:0]quant_in;
input [5:0]sel_in;
input [15:0]resolution;
input rstb;
output[5:0]quant_out;
output[15:0]fb;
wire [15:0]resolution;
wire [15:0]resolution_n = ~(resolution)+1;
wire [15:0]half_res = resolution/2;
reg  [15:0]fb;
reg  [5:0]quant_out;
wire [15:0] fb0 = 16'd24;
wire [15:0] fb1 = 16'd8;
wire [15:0] fb2 = -16'd8;
wire [15:0] fb3 = -16'd24;

always @(rstb or quant_in)
if (!rstb) begin
    fb = 16'b0000000000001000;
    quant_out  = 6'd0+sel_in;
end
else begin
    fb = quant_in[15] ? (quant_in >= resolution_n ? fb1 : fb0) :
                (quant_in >= resolution ? fb3 : fb2) ;
    quant_out  = quant_in[15] ? (quant_in >= resolution_n ? (6'd1+sel_in) : (6'd0+sel_in)) :
                (quant_in >= resolution ? (6'd3+sel_in) : (6'd2+sel_in)) ;
end
endmodule

//////////////////////////////////////////////////////////////
//     Pseudo random noise generator                      //
//////////////////////////////////////////////////////////////
module prn_generator(clk,rstb,prn);
input clk;
input rstb;
output prn;
reg [24:0]regis; //25 registers for the shift register
reg g25;         //generate term g25 for the polynomial
wire prn = regis[0];

/// the polynomial is {z^25+Z^22+1} i.e g25=g22^g0.
//initial state is 25'b1;

always @ (posedge clk or negedge rstb)
begin
  if(!rstb)begin
   regis=25'b0000000000000000000000001;
 end else begin
   g25=regis[22]+regis[0];
   regis={g25,regis[24:1]};
 end
end

endmodule
```

```
////////////////////////////////////////////////////////////////
//        Delta sigma modulator 2nd order                   //
////////////////////////////////////////////////////////////////

 module del_sig_mod2(mod_in,sel_in,resolution,rstb,clk,mod_out);
  input [15:0]mod_in;
  input [5:0]sel_in;
  input [15:0]resolution;
  input rstb;
  input clk;
  output[5:0]mod_out;


  wire prn;
  wire   [15:0] acc1_in, acc2_in, acc1_out, acc2_out, quant_in, fb,prn_gain;

  reg    [5:0]  mod_out;
  wire    [5:0]  mod_out_temp;
  wire [15:0]resolution_n = ~(resolution)+1;
  assign prn_gain= prn?16'd4:-16'd4;

always @(posedge clk or negedge rstb)
  if(!rstb) begin
    mod_out=6'b00_0010;
  end else begin
    mod_out=mod_out_temp;
end

always @(posedge clk)
   $display ("(%0t) DS => pn_out: %0d res: %0d mod_in: %0d fb: %0d acc1_in: %0d acc1_out: %0d
quant_in: %0d quant_out: %0d mod_out:%b ",
              $time, $signed(prn_gain), resolution, $signed(mod_in), $signed(fb),
              $signed(acc1_in), $signed(acc1_out), $signed(quant_in), mod_out_temp, mod_out);

  cla16 A1        (.a(mod_in)  ,.b(fb),.cin(1'b0),.sum(acc1_in),.cout16()); //fb inverted and cin=1 for 2's compl
  accumulator1 ac1 (.x(acc1_in), .clk(clk), .rstb(rstb),.y(acc1_out));
  cla16 A2        (.a(acc1_out),.b(fb),.cin(1'b0),.sum(acc2_in),.cout16());//fb inverted and cin=1 for 2's compl
  accumulator2 ac2 (.x(acc2_in), .clk(clk), .rstb(rstb),.y(acc2_out));
  prn_generator prn1 (.clk(clk),.rstb(rstb),.prn(prn));
  cla16 A3        (.a(acc2_out),.b(prn_gain),.cin(1'b0),.sum(quant_in),.cout16());
  quantizer quant1
(.quant_in(quant_in),.sel_in(sel_in),.resolution(resolution),.rstb(rstb),.quant_out(mod_out_temp),.fb(fb));
  endmodule
```

Digital calibration FSM with SPI verilog code

```
//Verilog HDL for "digital_calib_blocks", "digital_calibration" "verilog"
`timescale 1ns/100ps

/*********************************************************************************************
                                            SPI Slave
*********************************************************************************************/
module spi_slave_new_protocol_comp(rstb, SCK, SIMO, SOMI, SS_BAR, address, data_in,
                                                   data_out,read_write_bar,write_strobe);
input rstb;
input SCK;
input SIMO;
```

```
output SOMI;
input SS_BAR;
output [14:0] address;
output [7:0] data_in;
input [7:0] data_out;
output read_write_bar;
output write_strobe;

reg [7:0] data_in;
reg [14:0] shift_in;
reg [7:0] shift_out;
reg [4:0] bit_count;

wire read_strobe;
wire write_strobe;
wire [7:0] data_out;

reg [14:0] address;
reg read_write_bar;
reg shifted;

//bit count counts the no of bits shifted in or posedge SS_BAR
always@(negedge rstb or negedge SCK or posedge SS_BAR) begin
        if(~rstb)
                        bit_count <=  0;
        else if(SS_BAR)
                        bit_count <=  0;
        else if(~SS_BAR && shifted)
                        bit_count <=  bit_count + 5'd1;
        else
                bit_count <=  bit_count;
end


//shift register
always@(negedge rstb or posedge SCK) begin
        if(~rstb)
                begin
                        shift_in <=  0;
                        shifted <=  0;
                end
        else if(~SS_BAR)
                begin
                        shift_in <=  {shift_in[14:0], SIMO};
                        shifted <=  1;
                end
        else
                begin
                        shift_in <=  shift_in;
                        shifted <=  0;
                end
end

//this is for reading the data
always@(negedge rstb or negedge SCK) begin
        if(~rstb)
                shift_out <=  8'h00;
        else if(SS_BAR)
                shift_out <=  8'h00; //do not shift when SS_BAR deasserted
        else if(bit_count == 15 && read_write_bar)
                shift_out <=  data_out;
        else
                shift_out <=  shift_out << 1;
```

```
end


bufif1 b1(SOMI, shift_out[7], ~SS_BAR);

//address register

always@(negedge rstb or posedge SCK) begin
if(~rstb)
            address <=  0;
else if(bit_count == 5'd15)
            address <=  {shift_in[13:0], SIMO};
else
            address <=  address;
end

//data register
always@(negedge rstb or posedge SCK) begin
if(~rstb)
            data_in <=  8'b0;
else if(bit_count == 5'd23 && ~read_write_bar)
            data_in <=  {shift_in[6:0], SIMO};
else
            data_in <=  data_in;
end



//read_write_bar-decides if it is a read or write
always@(negedge rstb or posedge SCK) begin
if(~rstb)
            read_write_bar <=  0;
else if(bit_count == 15) //saves read_write_bar at the second earliest time
            read_write_bar <=  shift_in[14];
else if(bit_count == 0) //reset the read_write_bar
            read_write_bar <=  0;
else
            read_write_bar <=  read_write_bar;
end

assign write_strobe = (bit_count == 5'd23) & ~read_write_bar;
//assign read_strobe = (bit_count == 5'd15) & read_write_bar ;
endmodule

/*********************************************************************************************
                                    TRANSMITTER REGISTERS AND FSM
*********************************************************************************************/
module trans_reg_fsm_comp( rstb, SCK, address, data_in, data_out,
read_write_bar,write_strobe,clk,sd_out,sel_out,dc_in,
dc_tran_in,rst_sar,rst_tran_sar,rst_sd,delay1_code,delay2_code,delay_tran_code,mask,shape);
//SERIAL INTERFACE PORT DECLARATIONS
input rstb;
input SCK;
input [14:0] address; //15 bits of addressable 8 bit registers
input [7:0] data_in; //stores data to be written
output [7:0] data_out; //stores data to be loaded to shift register
input read_write_bar;
input write_strobe;

//FSM INTERFACEport declarations
 //rstb ALREADY DECLARED
input clk;
input [6:0] dc_in;
input [6:0] dc_tran_in;
output [15:0]sd_out;
output [7:0]sel_out;
output rst_sar;
output rst_tran_sar;
output rst_sd; //active low
output [6:0]delay1_code;
output [6:0]delay2_code;
output [6:0] delay_tran_code;
output [31:0]mask;
output [83:0]shape;


//registers for serial interface
reg [7:0] data_out;

//registers for FSM output
reg [15:0]sd_out;
reg [7:0]sel_out;
reg rst_sar;
reg rst_tran_sar;
reg rst_sd;     //active low reset
reg [6:0]delay1_code;
reg [6:0]delay2_code;
reg [6:0] delay_tran_code;

//32 bit mask register to control the pulse width
reg [31:0]mask;

//84 bit shape register to control the pulse shape
reg [83:0]shape;

//FSM registers and variables
reg [2:0] state;
reg [2:0] next_state;
reg [7:0]index;
reg rst_cnt;
reg  state3_4;
reg  state4_5;
reg  state5_6;
reg[15:0] count_clk;

//register files
reg [7:0] disable_calib_fc_reg;//enable/disable centre freq. calib.
reg [7:0] sd_in_cal_fc1;//sd input for 3.5g calib.
reg [7:0] sd_in_cal_fc2;//sd input for 4g calib.
reg [7:0] sd_in_cal_fc3;//sd input for 4.5g calib.
reg [7:0] sel_in_cal_fc1;//add input(selects the phase group) for 3.5g calib.
reg [7:0] sel_in_cal_fc2;//add input(selects the phase group) for 4g calib.
reg [7:0] sel_in_cal_fc3;//add input(selects the phase group) for 4.5g calib.
reg [7:0] dc_out_fc1;//calibrated 3.5g fc code
reg [7:0] dc_out_fc2;//calibrated 3.5g fc code
reg [7:0] dc_out_fc3;//calibrated 3.5g fc code
reg [7:0] calib_start_reg;//initates calibration
reg [7:0] calib_status_reg;//records calib. status
reg [7:0] calib_count_reg;//counter for number of input calibrations
reg [7:0] enable_dc_fc_calib;//enable/disable delay/fc calibration
reg [7:0] delay1_offset_addr;//select the out for delay 1 for BF subsystem
reg [7:0] delay2_offset_addr;//select the out for delay 2 for BF subsystem
reg [7:0] centre_freq; //sets the centre freq to 3.5,4 or 4.5GHz
reg [7:0] wait_calib_sd_cycle_l;//time for sd dll to settle
reg [7:0] wait_calib_sd_cycle_m;//time for sd dll to settle
reg [7:0] wait_calib_sar_cycle_l;//time for sar dll to settle
reg [7:0] wait_calib_sar_cycle_m;//time for sar dll to settle
reg [7:0] bypass_calib_reg;//bypass calibration in case of problem
reg [7:0] delay1_byp;//code for delay 1 when by passed
reg [7:0] delay2_byp;//code for delay 2 when by passed
reg [7:0] tran_fc_code_byp;//centre. freq code when by passed

reg [7:0]sd_in_arr[0:31];//sd dll modulator inputs
reg [7:0]sel_in_arr[0:31];//phase group selection inputs
reg [7:0]dc_out_arr[0:31];//calibrated digital code word output

//32-bit mask registers

reg [7:0]mask0;
reg [7:0]mask1;
reg [7:0]mask2;
reg [7:0]mask3;

reg [7:0]shape0;
reg [7:0]shape1;
reg [7:0]shape2;
reg [7:0]shape3;
reg [7:0]shape4;
reg [7:0]shape5;
reg [7:0]shape6;
reg [7:0]shape7;
reg [7:0]shape8;
reg [7:0]shape9;
reg [7:0]shape10;


//internal variables
reg [15:0]wait_sd_cycle;
reg [15:0]wait_sar_cycle;
reg [15:0]wait_record_cycle;
reg[7:0] delay1_tmp;
reg[7:0] delay2_tmp;
reg [7:0]temp_ip;
reg dc_fcbar_cycle;
reg [7:0] calib_count;



///////////////////////-----------Serial interface starts here----------------////

//SERIAL REGISTERS DEFAULT VALUES
parameter  disable_calib_fc_reg_default = 8'h05;
parameter  sd_in_cal_fc1_default        = 8'h06;
parameter  sd_in_cal_fc2_default        = 8'hfe;
parameter  sd_in_cal_fc3_default        = 8'hf8;
parameter  sel_in_cal_fc1_default       = 8'h03;
parameter  sel_in_cal_fc2_default       = 8'h05;
parameter  sel_in_cal_fc3_default       = 8'h07;
parameter  dc_out_fc1_default          = 8'h00;
parameter  dc_out_fc2_default          = 8'h00;
parameter  dc_out_fc3_default          = 8'h00;
parameter  calib_start_reg_default      = 8'h00;
parameter  calib_status_reg_default     = 8'h00;
parameter  calib_count_reg_default      = 8'h02;
parameter  enable_dc_fc_calib_default   = 8'h03;
parameter  delay1_offset_addr_default   = 8'h00;
parameter  delay2_offset_addr_default   = 8'h01;
parameter centre_freq_default          = 8'h01;
parameter wait_calib_sd_cycle_l_default = 8'h60;
parameter wait_calib_sd_cycle_m_default = 8'h09;
parameter wait_calib_sar_cycle_l_default= 8'hc8;
parameter wait_calib_sar_cycle_m_default= 8'h00;
parameter bypass_calib_reg_default       = 8'h00;
parameter delay1_byp_default             = 8'h00;
parameter delay2_byp_default             = 8'h00;
parameter tran_fc_code_byp_default       = 8'h00;

//write into register
always@(negedge rstb or negedge SCK) begin
              if(~rstb) begin
   //reset all registers to default value;
     disable_calib_fc_reg[7:0] <=disable_calib_fc_reg_default;
     sd_in_cal_fc1[7:0]         <= sd_in_cal_fc1_default;
     sd_in_cal_fc2[7:0]         <= sd_in_cal_fc2_default;
     sd_in_cal_fc3[7:0]         <= sd_in_cal_fc3_default;
     sel_in_cal_fc1[7:0]        <= sel_in_cal_fc1_default;
     sel_in_cal_fc2[7:0]        <= sel_in_cal_fc2_default;
     sel_in_cal_fc3[7:0]        <= sel_in_cal_fc3_default;
     calib_start_reg[7:0]           <= calib_start_reg_default ;
     calib_count_reg[7:0]           <= calib_count_reg_default;
     enable_dc_fc_calib[7:0]    <= enable_dc_fc_calib_default;
     delay1_offset_addr[7:0]    <= delay1_offset_addr_default;
     delay2_offset_addr[7:0]    <= delay2_offset_addr_default;
     centre_freq[7:0]           <= centre_freq_default           ;
     wait_calib_sd_cycle_l[7:0] <= wait_calib_sd_cycle_l_default ;
     wait_calib_sd_cycle_m[7:0] <= wait_calib_sd_cycle_m_default ;
     wait_calib_sar_cycle_l[7:0]<= wait_calib_sar_cycle_l_default;
     wait_calib_sar_cycle_m[7:0]<= wait_calib_sar_cycle_m_default;
     bypass_calib_reg[7:0]      <= bypass_calib_reg_default       ;
     delay1_byp[7:0]            <= delay1_byp_default             ;
     delay2_byp[7:0]            <= delay2_byp_default             ;
     tran_fc_code_byp[7:0]          <= tran_fc_code_byp_default       ;
     /////////////////initialize arrays/////////////

     //delay code inputs
     sd_in_arr[0]<=8'h00;
     sd_in_arr[1]<=8'hfc;
     sel_in_arr[0]<=8'h04;
     sel_in_arr[1]<=8'h04;

     mask0 <=8'hff;
     mask1 <=8'hff;
     mask2 <=8'h00;
     mask3 <=8'h00;


     shape0<=8'hb6;
     shape1<=8'hdd;
     shape2<=8'hb6;
     shape3<=8'h24;
     shape4<=8'hb9;
     shape5<=8'h6d;
     shape6<=8'h92;
     shape7<=8'h94;
     shape8<=8'h24;
     shape9<=8'h00;
     shape10<=8'h00;



              end else if((write_strobe))begin
                 case(address)
                  15'd7   : disable_calib_fc_reg[7:0] <= #1 data_in;
```

```
                15'd8   : sd_in_cal_fc1[7:0]         <= #1 data_in;
     15'd9   : sd_in_cal_fc2[7:0]         <= #1 data_in;
     15'd10  : sd_in_cal_fc3[7:0]         <= #1 data_in;
     15'd11  : sel_in_cal_fc1[7:0]        <= #1 data_in;
     15'd12  : sel_in_cal_fc2[7:0]        <= #1 data_in;
     15'd13  : sel_in_cal_fc3[7:0]        <= #1 data_in;
     15'd17  : calib_start_reg[7:0]       <= #1 data_in;
     15'd19  : calib_count_reg[7:0]       <= #1 data_in;
     15'd20  : enable_dc_fc_calib[7:0]    <= #1 data_in;
     15'd21  : delay1_offset_addr[7:0]    <= #1 data_in;
     15'd22  : delay2_offset_addr[7:0]    <= #1 data_in;
     15'd23  : centre_freq[7:0]           <= #1 data_in;
     15'd24  : wait_calib_sd_cycle_l[7:0] <= #1 data_in;
     15'd25  : wait_calib_sd_cycle_m[7:0] <= #1 data_in;
     15'd26  : wait_calib_sar_cycle_l[7:0]<= #1 data_in;
     15'd27  : wait_calib_sar_cycle_m[7:0]<= #1 data_in;
     15'd28  : bypass_calib_reg[7:0]      <= #1 data_in;
     15'd29  : delay1_byp[7:0]            <= #1 data_in;
     15'd30  : delay2_byp[7:0]            <= #1 data_in;
     15'd31  : tran_fc_code_byp[7:0]      <= #1 data_in;

     //sd_inputs – similar code is used for the remaining 30 registers
     15'd32  : sd_in_arr[0]<= #1 data_in;
     15'd33  : sd_in_arr[1]<= #1 data_in;


     //sel_inputs - similar code is used for the remaining 30 registers
     15'd64  : sel_in_arr[0]<= #1 data_in;
     15'd65  : sel_in_arr[1]<= #1 data_in;


     //mask registers
     15'd128 : mask0 <= #1 data_in;
     15'd129 : mask1 <= #1 data_in;
     15'd130 : mask2 <= #1 data_in;
     15'd131 : mask3 <= #1 data_in;

     //shape registers
     15'd132 :shape0 <= #1 data_in;
     15'd133 :shape1 <= #1 data_in;
     15'd134 :shape2 <= #1 data_in;
     15'd135 :shape3 <= #1 data_in;
     15'd136 :shape4 <= #1 data_in;
     15'd137 :shape5 <= #1 data_in;
     15'd138 :shape6 <= #1 data_in;
     15'd139 :shape7 <= #1 data_in;
     15'd140 :shape8 <= #1 data_in;
     15'd141 :shape9 <= #1 data_in;
     15'd142 :shape10 <= #1 data_in;

              endcase
             end
end
//------end of writing into register ----------////////

//reading from register
always@(posedge read_write_bar or posedge SCK)begin

             if(read_write_bar)
             begin
                          case(address)
     15'd7   : data_out <= disable_calib_fc_reg[7:0];
     15'd8   : data_out <= sd_in_cal_fc1[7:0];
```

```
15'd9   : data_out <= sd_in_cal_fc2[7:0];
15'd10  : data_out <= sd_in_cal_fc3[7:0];
15'd11  : data_out <= sel_in_cal_fc1[7:0];
15'd12  : data_out <= sel_in_cal_fc2[7:0];
15'd13  : data_out <= sel_in_cal_fc3[7:0];
15'd14  : data_out <= dc_out_fc1[7:0];
15'd15  : data_out <= dc_out_fc2[7:0];
15'd16  : data_out <= dc_out_fc3[7:0];
15'd17  : data_out <= calib_start_reg[7:0];
15'd18  : data_out <= calib_status_reg[7:0];
15'd19  : data_out <= calib_count_reg[7:0];
15'd20  : data_out <= enable_dc_fc_calib[7:0];
15'd21  : data_out <= delay1_offset_addr[7:0];
15'd22  : data_out <= delay2_offset_addr[7:0];
15'd23  : data_out <= centre_freq[7:0];
15'd24  : data_out <=  wait_calib_sd_cycle_l[7:0];
15'd25  : data_out <= wait_calib_sd_cycle_m[7:0];
15'd26  : data_out <= wait_calib_sar_cycle_l[7:0];
15'd27  : data_out <=  wait_calib_sar_cycle_m[7:0];
15'd28  : data_out <= bypass_calib_reg[7:0];
15'd29  : data_out <=  delay1_byp[7:0];
15'd30  : data_out <=  delay2_byp[7:0];
15'd31  : data_out <= tran_fc_code_byp[7:0];


//sd_inputs similar code is used for the remaining 30 registers
15'd32  : data_out <= sd_in_arr[0];
15'd33  : data_out <= sd_in_arr[1];


//sel_inputs -similar code is used for the remaining 30 registers
15'd64  : data_out <= sel_in_arr[0];
15'd65  : data_out <= sel_in_arr[1];


//dc_output -  similar code is used for the remaining 30 registers
15'd96  : data_out  <= dc_out_arr[0];
15'd97  : data_out  <= dc_out_arr[1];

//mask registers
15'd128 : data_out  <= mask0;
15'd129 : data_out  <= mask1;
15'd130 : data_out  <= mask2;
15'd131 : data_out  <= mask3;

//shape registers
15'd132 :data_out  <= shape0;
15'd133 :data_out  <= shape1;
15'd134 :data_out  <= shape2;
15'd135 :data_out  <= shape3;
15'd136 :data_out  <= shape4;
15'd137 :data_out  <= shape5;
15'd138 :data_out  <= shape6;
15'd139 :data_out  <= shape7;
15'd140 :data_out  <= shape8;
15'd141 :data_out  <= shape9;
15'd142 :data_out  <= shape10;


           default : data_out  <= 0;
                      endcase
        end else
        begin
```

```
                data_out<=data_out;    //removing this line will hold the data on to the bus
            end
end//always
///end of reading from register


 ////////////////-------------------END OF serial interface--------////////////////////

 ///////////////-------------------FSM starts -------------------////////////////////




 //FSM STATES
parameter  [2:0]  INITIALIZE          =3'b000;
parameter  [2:0]  START_CALIB       =3'b001;
parameter  [2:0]  LOAD_IP              =3'b010;
parameter  [2:0]  START_SAR         =3'b011;
parameter  [2:0]  RECORD_DATA   =3'b100;
parameter  [2:0]  DEC_COUNT        =3'b101;
parameter  [2:0]  WAIT1                 =3'b110;
integer i;

always@(posedge clk)begin
dc_fcbar_cycle=calib_status_reg[0] & calib_status_reg[1] & calib_status_reg[2];
end

always@(posedge clk)begin
wait_sd_cycle<={wait_calib_sd_cycle_m,wait_calib_sd_cycle_l};
wait_sar_cycle<= {wait_calib_sd_cycle_m,wait_calib_sd_cycle_l}+
{wait_calib_sar_cycle_m,wait_calib_sar_cycle_l};
wait_record_cycle<={wait_calib_sd_cycle_m,wait_calib_sd_cycle_l}+
{wait_calib_sar_cycle_m,wait_calib_sar_cycle_l}+15'd10;
end

always@( negedge rstb or posedge clk)begin
            if(~rstb)
            begin
                        calib_count<=0;
                        index<=0;
                        calib_status_reg<=8'h00;
            end
            else if(~calib_start_reg[0])
            begin
                        index<=0;
                          calib_count<= calib_count_reg;
                          calib_status_reg[0]<=1'b0;
                          calib_status_reg[1]<=1'b0;
                          calib_status_reg[2]<=1'b0;
                          calib_status_reg[3]<=1'b0;
                        if(disable_calib_fc_reg[0])
                          calib_status_reg[0]<=1'b1;
                        if(disable_calib_fc_reg[1])
                          calib_status_reg[1]<=1'b1;
                        if(disable_calib_fc_reg[2])
                          calib_status_reg[2]<=1'b1;

            end
            else if(state==DEC_COUNT)
            begin
                if(~dc_fcbar_cycle & enable_dc_fc_calib[0])
                          begin
                                          calib_count<=calib_count;
                          end
                          else if(enable_dc_fc_calib[1])
                          begin
                                          index<= index+1;
                                          calib_count <= calib_count-1;
                          end
                          else
                          begin
                        calib_count<=calib_count;
       end
       ////////////////////
                          if(~dc_fcbar_cycle & enable_dc_fc_calib[0])begin
                                          if(~calib_status_reg[0])begin
                                          calib_status_reg[0]<=1'b1;
                                          end else if (calib_status_reg[0] & ~calib_status_reg[1])begin
                                          calib_status_reg[1]<=1'b1;
                                          end else if (calib_status_reg[0] & calib_status_reg[1] &
~calib_status_reg[2])begin
                                          calib_status_reg[2]<=1'b1;
                                          end
       end
            end//else if(state==DEC_COUNT) begin
            else if(state==WAIT1)
            begin
              if(calib_count==0)
                        begin
       calib_status_reg[3]<=1'b1;
                          calib_count<=calib_count;
                        end
   end
   else
            begin
                        calib_count<=calib_count;
            end


end



always@(negedge rst_cnt or posedge clk)begin
if(~rst_cnt)begin
   count_clk=16'd0;
   state3_4=0;
   state4_5=0;
   state5_6=0;

end else begin
   count_clk=count_clk+1;
if(count_clk > wait_sd_cycle)
   state3_4=1;
if(count_clk > wait_sar_cycle )
   state4_5=1;
if(count_clk > wait_record_cycle)
   state5_6=1;
end
end

//state machine uses sequential logic (blocking statements)
always@(state or calib_start_reg[0] or dc_fcbar_cycle or calib_status_reg[3] or calib_status_reg[0] or
calib_status_reg[1] or calib_status_reg[2] or state3_4 or   state4_5  or state5_6  )begin
//state transistions
case(state)
```

```
   INITIALIZE:
   begin
      next_state=START_CALIB;
   end
   START_CALIB  : begin
    if(calib_start_reg[0] ==1'b1) begin
         if((~dc_fcbar_cycle & enable_dc_fc_calib[0]) || (calib_count>0 & enable_dc_fc_calib[1] &
~calib_status_reg[3]))begin
            next_state=LOAD_IP;
         end else begin
           next_state=START_CALIB;
         end
      end else begin
    next_state=START_CALIB;
    end
   end

   LOAD_IP  :begin
      if(state3_4)begin
      next_state=START_SAR;
      end else  begin
      next_state=LOAD_IP;
      end
  end



   START_SAR: begin
      if(state4_5)begin
      next_state=  RECORD_DATA;
      end else begin
      next_state=START_SAR;
      end
  end
   RECORD_DATA :begin
      if(state5_6)begin
      next_state=DEC_COUNT;
      end else begin
      next_state=  RECORD_DATA;
      end
  end


DEC_COUNT: begin
          next_state=WAIT1;
end

WAIT1:next_state=START_CALIB;
endcase

end





always@(negedge rstb or posedge clk)begin
//signal transistions
              if(~rstb)begin
                            state<=INITIALIZE;
              end else begin
                            state <=next_state;
              end
```

```
end

//combinational logic so use non blocking statements
always@(state)begin
//it is always better to create a neg edge on rst_cnt to reset it properly
case (state)
  INITIALIZE:
  begin
      rst_sd=1'b1;
      sd_out=16'd0;
      sel_out=8'd0;
  end

  START_CALIB:begin
      rst_cnt=1'b0;
      rst_sd=1'b0;
      rst_sar=1'b0;
      rst_tran_sar=1'b0;
  end
  LOAD_IP:
  begin
      rst_cnt=1'b1;
      rst_sd=1'b0;
      if(~dc_fcbar_cycle & enable_dc_fc_calib[0])begin
         if(~calib_status_reg[0])begin
         ////////////////////////////////////////
            if(sd_in_cal_fc1[7])
            sd_out={8'b1111_1111,sd_in_cal_fc1};
            else
            sd_out={8'b0000_0000,sd_in_cal_fc1};
         ////////////////////////////////////////
         sel_out=sel_in_cal_fc1;
         rst_tran_sar=1'b0;
         rst_sd=1'b1;
         end else if (calib_status_reg[0] & ~calib_status_reg[1])begin
          ////////////////////////////////////////
            if(sd_in_cal_fc2[7])
            sd_out={8'b1111_1111,sd_in_cal_fc2};
            else
            sd_out={8'b0000_0000,sd_in_cal_fc2};
         ////////////////////////////////////////
         sel_out=sel_in_cal_fc2;
         rst_tran_sar=1'b0;
         rst_sd=1'b1;
         end else if (calib_status_reg[0] & calib_status_reg[1] & ~calib_status_reg[2])begin
         ////////////////////////////////////////
            if(sd_in_cal_fc3[7])
            sd_out={8'b1111_1111,sd_in_cal_fc3};
            else
            sd_out={8'b0000_0000,sd_in_cal_fc3};
         ////////////////////////////////////////

         sel_out=sel_in_cal_fc3;
         rst_tran_sar=1'b0;
         rst_sd=1'b1;
         end
    end else begin

         ////////////////////////////////////////
            temp_ip=sd_in_arr[index];
            if(temp_ip[7])
            sd_out={8'b1111_1111,sd_in_arr[index]};
            else
          sd_out={8'b0000_0000,sd_in_arr[index]};
        //////////////////////////////////////

        sel_out=sel_in_arr[index];
        rst_cnt=1;
        rst_sar=0;
        rst_sd=1;
    end

  end
  START_SAR:begin
     if(~dc_fcbar_cycle & enable_dc_fc_calib[0])begin
      rst_tran_sar=1'b1;
      end else begin
      rst_sar=1'b1;
      end

  end
  RECORD_DATA:begin
      if(~dc_fcbar_cycle & enable_dc_fc_calib[0])begin

        if(~calib_status_reg[0])begin
        dc_out_fc1={1'b0,dc_tran_in};
        end else if (calib_status_reg[0] & ~calib_status_reg[1])begin
        dc_out_fc2={1'b0,dc_tran_in};
        end else if (calib_status_reg[0] & calib_status_reg[1] & ~calib_status_reg[2])begin
        dc_out_fc3={1'b0,dc_tran_in};
        end
     end else begin
     dc_out_arr[index]={1'b0,dc_in};
     end

   end
  DEC_COUNT:begin
      rst_sd=1'b0;
end
endcase
end

////-----------------------End of FSM ---------------------------///////////


always@(posedge clk)begin
   if(~bypass_calib_reg[0]) begin
    delay1_tmp<=dc_out_arr[delay1_offset_addr];
    delay2_tmp<=dc_out_arr[delay2_offset_addr];
   end else begin
    delay1_tmp<=delay1_byp;
    delay2_tmp<=delay2_byp;
   end
end

always@(posedge clk)begin
    delay1_code<=delay1_tmp[6:0];
    delay2_code<=delay2_tmp[6:0];
    mask={mask3,mask2,mask1,mask0};
    shape={shape10[3:0],shape9,shape8,shape7,shape6,shape5,shape4,shape3,shape2,shape1,shape0};
end


always@(posedge clk)begin
   if(~bypass_calib_reg[0])begin
      case(centre_freq[1:0])
```

```
         2'b00:delay_tran_code<=dc_out_fc1[6:0];
         2'b01:delay_tran_code<=dc_out_fc2[6:0];
         2'b10:delay_tran_code<=dc_out_fc3[6:0];
         2'b11:delay_tran_code<=dc_out_fc1[6:0];
         default:delay_tran_code<=dc_out_fc1[6:0];
      endcase
   end else begin
   delay_tran_code<=tran_fc_code_byp[6:0];
   end
end


endmodule

/*******************************************************************************************
             DIGITAL CALIBRATION
*******************************************************************************************/

module digital_calibration_comp(SCK,SS_BAR,SIMO,SOMI,clk,dc_in,dc_tran_in,sd_out,sel_out,
rst_sar,rst_tran_sar,rst_sd,delay1_code,delay2_code,
delay_tran_code,resolution,rstb,mask,shape);

//serial interface
input SCK;
input SS_BAR;
input SIMO;
output SOMI;

//parallel calibration interface
input clk;
input [6:0] dc_in;
input [6:0] dc_tran_in;
output [15:0]sd_out;
output [5:0]sel_out;
output rst_sar;
output rst_tran_sar;
output rst_sd;
output [6:0]delay1_code;
output [6:0]delay2_code;
output [6:0] delay_tran_code;
output [15:0] resolution;
output [31:0]mask;
output [83:0]shape;

input rstb;
wire [15:0] resolution;
wire SCK;
wire SIMO;
wire SOMI;
wire SS_BAR;
wire clk;
wire [14:0] address;
wire [7:0] data_out;
wire[7:0] data_in;
wire write_strobe;
wire [6:0] dc_in;
wire [6:0] dc_tran_in;
wire [15:0]sd_out;
wire [5:0]sel_out;
wire rst_sar;
wire rst_tran_sar;
wire rst_sd;
wire[6:0]delay1_code;
```

```
wire [6:0]delay2_code;
wire [6:0] delay_tran_code;
wire rstb;
wire [7:0]sel_out_temp;
wire [31:0]mask;
wire [83:0]shape;

assign sel_out= sel_out_temp[5:0];
assign resolution=16'd16;

spi_slave_new_protocol_comp s1(rstb, SCK, SIMO, SOMI, SS_BAR, address, data_in,
                                        data_out,read_write_bar,write_strobe);


trans_reg_fsm_comp t1( .rstb(rstb),
        .SCK(SCK),
        .address(address),
        .data_in(data_in),
        .data_out(data_out),
        .write_strobe(write_strobe),
        .read_write_bar(read_write_bar) ,
        .clk(clk),
        .sd_out(sd_out),
        .sel_out(sel_out_temp),
        .dc_in(dc_in),
        .dc_tran_in(dc_tran_in),
        .rst_sar(rst_sar),
        .rst_tran_sar(rst_tran_sar),
        .rst_sd(rst_sd),
        .delay1_code(delay1_code),
        .delay2_code(delay2_code),
        .delay_tran_code(delay_tran_code),
        .mask(mask),
        .shape(shape));


endmodule
```