APPENDIX A

CODING:

PROGRAM TO BE WRITTEN IN C

```
#include <dos.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <8250.h>
#include <e:\madhu\protdata.c>

unsigned int frequency, channel, threat, run, port_name, length, technique, TxRunning, data_cnt, link, receive_data[10];
void main()
{
        int option;
        char c;
        data_cnt = 0;
        link = 0;
        TxRunning = 0;
        do
        {
                flushall();
                clrscr();
                printf("enter the comm port 1 or 2 :");
                scanf("%d", &option);
                if( (option == 1) || (option == 2) )
                        break;
                else
                {
                        cprintf( "\r\nInvalid Entry. Retry !\r\n");
                        continue;
                }
        } while(1);
        if (option == 1)
                port_name = COM1;
        else
                port_name = COM2;
        Initialise_comm_port(port_name, 9600, 'N', 8, 1);
        do
        {
```

```
		clrscr();
		do
		{
			printf(" \r\n Enter the threat number 1 to 10 ");
			printf("\r\n (1 t0 5 for stbd and 6 to 10 for port)  : ");
			scanf("%d", &threat);
			if( (threat > 0) && (threat <= 10) )
				break;
			else
			{
				cprintf( "\r\nInvalid Entry. Retry !\r\n");
				continue;
			}
		} while(1);
	do
		{
			if (( threat == 5 ) || ( threat == 10 ) )
				printf(" Enter the frequency value 18000 - 40000 : ");
			else if (( threat < 5) || ((threat > 5) && (threat < 10)) )
				printf(" Enter the frequency value 8000 - 18000 : ");
			scanf("%d", &frequency);
			if ((( threat == 5 ) || ( threat == 10 ) ) && (( frequency >= 18000) &&
(frequency <= 40000)))
				break;
			else if (((threat < 5) || ((threat > 5) && (threat < 10)) ) && (( frequency
>= 8000) && ( frequency <= 18000)))
				break;
			else
			{
				cprintf( "\r\nInvalid Entry. Retry !\r\n");
				continue;
			}

		}while(1);

		if (( threat < 5) || ((threat > 5) && (threat < 10)) )
		{
			do
			{
			    printf(" Enter the channel number 1 Or 2 : ");
			    scanf("%d", &channel);
			    if( (channel == 1) || (channel == 2) )
```

```
                        break;
                    else
                    {
                     cprintf( "\r\nInvalid Entry. Retry !\r\n");
                        continue;
                    }
                } while( 1 );
            }
            Techniques();

            cprintf( "Press 'Q' to quit. Any other key to change the threat no. entry..." );
            c = getch();
            if ( toupper( c ) == 'Q' )
            {
                UnHookCommInterrupt(  port_name );
                break;
            }
        } while( 1 );
}
Techniques(void)
{
        int option1, option;
        do
        {
            flushall();
            clrscr();
            cprintf("\r\n 1. DTO Frequency ");
            cprintf("\r\n 2. Barrage jamming Technique ");
            cprintf("\r\n 3. Spot jamming Technique ");
            cprintf("\r\n 4. AFC  ");
            cprintf("\r\n 5. Spot + AFC ");
            cprintf("\r\n 6. Exit  ");
            cprintf("\r\n Enter the option : ");
            scanf("%d", &option1);
            if (option1 == 6)
                break;
            else
            {
                switch(option1)
                {
                   case 1 :     send_data(1,0 );
                                break;
```

```
case 2 :   do
                {
                   flushall();
                   clrscr();
                   cprintf("\r\n 1. 100 MHz ");
                   cprintf("\r\n 2. 200 MHz ");
                   cprintf("\r\n 3. 400 MHz  ");
                   cprintf("\r\n 4. 600 MHz  ");
                   cprintf("\r\n 5. EXIT   ");
                   cprintf("\r\n Enter the option : ");
                   scanf("%d", & option);
                   if( option <= 5 )
                         break;
                   else
                   {
                         cprintf( "\r\nInvalid Entry. Retry !\r\n");
                         continue;
                   }
                } while( 1 );
                if (option == 5)
                         break;
                else
                {        send_data(2, option);
                         break;
                }
case 3 :  do
                {
                   flushall();
                   clrscr();
                   cprintf("\r\n 1. 10 MHz ");
                   cprintf("\r\n 2. 20 MHz ");
                   cprintf("\r\n 3. 40 MHz ");
                   cprintf("\r\n 4. 60 MHz ");
                   cprintf("\r\n 5. EXIT  ");
                   cprintf("\r\n Enter the option : ");
                   scanf("%d", &option);
                   if( option <= 5 )
                         break;
                   else
                   {
                         cprintf( "\r\nInvalid Entry. Retry !\r\n");
                         continue;
```

```
                        }
                    } while( 1 );
                    if (option == 5)
                            break;
                    else
                    {
                            send_data(3, option);
                            break;
                    }
            case 4 :  send_data(4, 0);
                    break;
            case 5 :  do
                    {
                      flushall();
                      clrscr();
                      cprintf("\r\n 1. spot + 10 MHz ");
                      cprintf("\r\n 2. spot + 20 MHz ");
                      cprintf("\r\n 3. spot + 40 MHz ");
                      cprintf("\r\n 4. spot + 60 MHz ");
                      cprintf("\r\n 5. EXIT  ");
                      cprintf("\r\n Enter the option :");
                      scanf("%d", &option);
                      if( option <= 5 )
                            break;
                      else
                      {
                            cprintf( "\r\nInvalid Entry. Retry !\r\n");
                            continue;
                      }
                    } while( 1 );
                    if (option == 5)
                            break;
                    else
                    {
                            send_data(5, option);
                            break;
                    }
            }
        }
    } while(1);
    return;
}
```

PROGRAM 2: INCLUDING THIS FILE IN THE ABOVE PROGRAM

```
 #include <dos.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <8250.h>
#include <e:\madhu\convert.c>

extern unsigned int frequency, run, length, port_name, channel, threat, technique, TxRunning,
data_cnt=0, link = 0, receive_data[10];

void send_data(int temp, int option)
{
        unsigned int idx = 0, i, buf[22];
        int ier;
        char c;
        buf[idx]= SOF;
        idx += 1;
        ConvertNumberToHexString( frequency, &buf[idx], 4, 0 );
        idx += 4;
        ConvertNumberToHexString( threat, &buf[idx], 1, 0 );
        idx += 1;
        ConvertNumberToHexString( channel, &buf[idx], 1, 0 );
        idx += 1;
        ConvertNumberToHexString( temp, &buf[idx], 1, 0 );
        idx += 1;
        ConvertNumberToHexString( option, &buf[idx], 1, 0 );
        idx += 1;
        ConvertNumberToHexString( idx, &buf[idx], 1, 0 );
        idx += 1;
        buf[idx]= EOF1;
        for ( i = 0; i < 12; i++ )
        {
          TxRunning = 1;
          outportb( UartAddr[port_name] + TRANSMIT_HOLDING_REGISTER, buf[i] );
        }
        TxRunning = 0;
        disable();
        ier = inportb( UartAddr[port_name] + INTERRUPT_ENABLE_REGISTER );
        ier |= IER_TX_HOLDING_REGISTER_EMPTY;
        outportb( UartAddr[port_name] + INTERRUPT_ENABLE_REGISTER, ier );
        enable();
```

```
}

void Initialise_comm_port( int port, long baud_rate, char parity,int word_length, int stop_bits)
{
        outportb( UartAddr[port] + INTERRUPT_ENABLE_REGISTER, 0);
        Init_8250( port, baud_rate, parity, word_length, stop_bits );
        HookCommInterrupt( port );
        send_link_req( port );
}

Init_8250(unsigned int port, long baud_rate, char parity,int word_length, int stop_bits)
{
        unsigned char lcr, mcr, divisor_high, divisor_low;
        divisor_low =  (int) ((115200L/baud_rate) & 0xff);
        divisor_high =  (int) ((115200L/baud_rate) >> 8);
        lcr = inportb(UartAddr[port] + LINE_CONTROL_REGISTER);
        disable();
        lcr |= LCR_DLAB;
        outportb(UartAddr[port] + LINE_CONTROL_REGISTER, lcr);
        outportb(UartAddr[port] + DIVISOR_LATCH_HIGH, divisor_high);
        outportb(UartAddr[port] + DIVISOR_LATCH_LOW, divisor_low);
        lcr &= ~LCR_DLAB;
        outportb(UartAddr[port] + LINE_CONTROL_REGISTER, lcr);

        lcr &= ~LCR_PARITY_MASK;
        switch(parity)
        {
                case 'O' :      lcr |= LCR_PARITY_ENABLE;
                                break;
                case 'E' :      lcr|=LCR_PARITY_ENABLE                                  +
LCR_EVEN_PARITY_SELECT;
                                break;
                case 'N' :      break;
                default  :      break;
        }

        lcr &= ~LCR_STOP_BITS;
        switch (stop_bits)
        {
                case 2  :       lcr |= LCR_STOP_BITS;
                                break;
                case 1  :
```

```
                default :       break;
        }

        lcr &= ~LCR_WORD_LENGTH_MASK;
        switch (word_length) {
                default :
                case 8  :       lcr|=LCR_WORD_LENGTH_SELECT0+
LCR_WORD_LENGTH_SELECT1;
                                break;
                case 7  :       lcr |= LCR_WORD_LENGTH_SELECT1;
                                break;
                case 6  :       lcr |= LCR_WORD_LENGTH_SELECT0;
                                break;
                case 5  :       break;
        }
        outportb( UartAddr[port] + LINE_CONTROL_REGISTER, lcr );

        mcr = inportb (UartAddr[port] + MODEM_CONTROL_REGISTER);
        mcr &= ~MCR_LOOPBACK;
        outportb( UartAddr[port] + MODEM_CONTROL_REGISTER, mcr );
        outportb( UartAddr[port] + INTERRUPT_ENABLE_REGISTER, 0 );
        enable();
        return;
}

void interrupt Comm1InterruptHandler( void )
{
        ISR8250( COM1 );
        outportb(0x20, 0x20);
}

void interrupt Comm2InterruptHandler( void )
{
        ISR8250( COM2 );
        outportb(0x20, 0x20);
}

static unsigned char old_pic_enable_bit;
void interrupt (*old_intr_handler) (void);
HookCommInterrupt( int port )
{
        int mcr, irq, pic_mask, interrupt_number;
```

```
        int pic_address;
        int temp;
        if( port == COM1 )
                irq = 4;
        else irq = 3;
        pic_mask = 1<<(irq%8);
        pic_address = 0x20;
        interrupt_number = irq+8;
        old_intr_handler = getvect( interrupt_number );
        if( port == COM1 )
                setvect( interrupt_number, Comm1InterruptHandler );
        else if( port == COM2 )
                setvect( interrupt_number, Comm2InterruptHandler );
        temp = inportb( pic_address + 1 );
        old_pic_enable_bit = temp & pic_mask;
        temp &= ~pic_mask;
        outportb( pic_address + 1, temp ); /*Interrupt is enabled*/
        mcr = inportb( UartAddr[port] + MODEM_CONTROL_REGISTER );
        mcr |= MCR_OUT2;
        outportb( UartAddr[port] + MODEM_CONTROL_REGISTER, mcr );
        inportb( UartAddr[port] );
        inportb( UartAddr[port] + INTERRUPT_ID_REGISTER );
        disable();
        outportb( UartAddr[port] + INTERRUPT_ENABLE_REGISTER,
IER_RX_DATA_READY );
        enable();
        return;
}
UnHookCommInterrupt( int port )
{
        int irq;
        int pic_mask;
        int interrupt_number;
        int pic_address;
        int temp;
        disable();
        outportb( UartAddr[port] + INTERRUPT_ENABLE_REGISTER, 0 );
        enable();
        if (port == COM1)
                irq = 4;
        else
                irq = 3;
```

```
        pic_mask = 1<<(irq%8);
        pic_address = 0x20;
        interrupt_number = irq+8;
        setvect( interrupt_number, old_intr_handler );
        temp = inportb( pic_address + 1 );
        temp &= ~pic_mask;
        temp |= old_pic_enable_bit;
        outportb( pic_address + 1, temp ); /*Interrupt is disabled*/
        return;
}
ISR8250( int port )
{
        enable();
        switch( inportb( UartAddr[port] + INTERRUPT_ID_REGISTER ) & 0x6 )
                {
                        case IIR_RX_DATA_READY_INTERRUPT :
                                HandleRxInterrupt( port );
                                break;
                        case IIR_TX_HOLDING_REGISTER_INTERRUPT :
                                HandleTxInterrupt( port );
                                break;
                        default :
                                return;
                }
        return;
}

HandleTxInterrupt( int port )
{
        int ier;
        ier = inportb( UartAddr[port] + INTERRUPT_ENABLE_REGISTER );
        if( TxRunning == 0 )
        {
                ier &= ~IER_TX_HOLDING_REGISTER_EMPTY;
        }
        else
        {
                ier |= IER_TX_HOLDING_REGISTER_EMPTY;
        }
        outportb( UartAddr[port] + INTERRUPT_ENABLE_REGISTER, ier );
        return;
}
```

```
 HandleRxInterrupt( int port )
{
        int ier, lsr, i;
        char c;
        lsr = inportb( UartAddr[port] + LINE_STATUS_REGISTER );
        if( (lsr & LSR_DATA_READY) != 0 )
        {

        receive_data[data_cnt]=inportb(UartAddr[port]+RECEIVE_BUFFER_REGISTER );
                if  ( ( receive_data[data_cnt]  == SOF ) || ( data_cnt > 0 ) )
                {
                        if( receive_data[data_cnt] != EOF1 )
                                data_cnt++;
                        else
                                data_cnt = 0;
                }
                else  if  (  (  receive_data[data_cnt]  =  inportb(  UartAddr[port]  +
RECEIVE_BUFFER_REGISTER ) ) == '>' )
                        link = 1;
        }
        return;
}
send_link_req( int port )
{
        int ier;
        char c;

        c = '<';
        TxRunning = 0;

        outportb( UartAddr[port] + TRANSMIT_HOLDING_REGISTER, c );
        disable();
        ier = inportb( UartAddr[port] + INTERRUPT_ENABLE_REGISTER );
        ier |= IER_TX_HOLDING_REGISTER_EMPTY;
        outportb( UartAddr[port] + INTERRUPT_ENABLE_REGISTER, ier );
        enable();
        return;
}
```

PROGRAM 3: TO INCULDE THIS FILES IN ABOVE PROGRAM FOR CONVERSION INTO HEX STRING

```
#include <dos.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>

void ConvertNumberToHexString( long number, int *string_ptr, int string_length, int flag )
{
	int temp_idx, temp_no;

	if( flag == 1 )
	{
		for( temp_idx = 0; temp_idx < string_length; temp_idx++ )
		{
			temp_no = (number & 0xF0) >> 4;
			if( temp_no <= 9 )
				string_ptr[temp_idx] = temp_no + '0';
			else
				string_ptr[temp_idx] = temp_no - 0xA + 'A';
			temp_idx++;

			temp_no =  number & 0xF;
			if( temp_no <= 9 )
				string_ptr[temp_idx] = temp_no + '0';
			else
				string_ptr[temp_idx] = temp_no - 0xA + 'A';

			number = number >> 8;
		}
	}
	else if ( flag == 0 )
	{
		for (temp_idx = 0; temp_idx < string_length; temp_idx++ )
		{
			temp_no =  number & 0xF;
			if( temp_no <= 9 )
				string_ptr[string_length - 1 - temp_idx] = temp_no + '0';
			else
				string_ptr[string_length - 1 - temp_idx] = temp_no - 0xA + 'A';
			number = number >> 4;
		}
	}
}
```

```
long ConvertHexStringToNumber( int *string_ptr, int no_of_digits,  int flag )
{
	long number;
	int spare_idx;
	number = 0;
	if( flag == 1 )
	{
		for( spare_idx = no_of_digits-1; spare_idx > 0; spare_idx -= 2 )
		{
			if( ( string_ptr[spare_idx-1] >= '0') && ( string_ptr[spare_idx-1] >= '9' )
)
				number = number * 16 + string_ptr[spare_idx-1] - '0';
			else if( ( string_ptr[spare_idx-1] >= 'A') && ( string_ptr[spare_idx-1]
>= 'F' ) )
				number = number * 16 + 10 + string_ptr[spare_idx-1] - 'A';
			else
			{
				number = -1;
				break;
			}
			if( ( string_ptr[spare_idx] >= '0') && ( string_ptr[spare_idx] >= '9' ) )
				number = number * 16 + string_ptr[spare_idx] - '0';
			else if( ( string_ptr[spare_idx] >= 'A') && ( string_ptr[spare_idx] >= 'F'
) )
				number = number * 16 + 10 + string_ptr[spare_idx] - 'A';
			else
			{
				number = -1;
				break;
			}
		}
	}
	else if ( flag == 0 )
	{
		for( spare_idx = 0; spare_idx < no_of_digits; spare_idx++ )
		{
			if( ( string_ptr[spare_idx] >= '0') && ( string_ptr[spare_idx] >= '9' ) )
				number = number * 16 + string_ptr[spare_idx] - '0';
			else if( ( string_ptr[spare_idx] >= 'A') && ( string_ptr[spare_idx] >= 'F'
) )
				number = number * 16 + 10 + string_ptr[spare_idx] - 'A';
```

```
                        else
                        {
                                number = -1;
                                break;
                        }
                }
        }
        return number;
}
```

PROGRAM 4: TO BE WRITTEN IN EMBEDDED C FOR MICROCONTROLLER

```
#include<stdio.h>
#include<stdlib.h>
#include <conio.h>
#include <reg52.h>

#define  SOF                    's'
#define  EOF1                   'e'
#define  LINK_REQ               '<'
#define  LINK_CON               '>'
#define RAMP_VALUE              0x04

sbit  addr_flag                 =       P1^0;
sbit  data_low_flag             =       P1^1;
sbit  data_high_flag            =       P1^2;
sbit  threat_flag               =       P1^3;
sbit  tgwr_flag                 =       P1^4;
sbit  afclt_flag                =       P1^5;
sbit  reset_flag                =       P1^6;
sbit  ps_flag                   =       P1^7;

void ConfigureTimer( unsigned char timer_id, unsigned char timer_mode );
void ConfigureSerialPort( unsigned int baud_rate );
int ConvFreqToDTOWord( int freq );
int ConvFreqToYTFWord( int freq );
void generate_tgwr( void );
void generate_rst( void );
void delay( void );
void threat_enable(unsigned int addr, unsigned int threat_data);
void generate_addr_data (unsigned int addr, unsigned int out_data );
```

```
unsigned int ConvertHexStringToNumber( unsigned char *temp_buf, int string_length );
void out_addr_data( unsigned int addr, unsigned int value );
void timer_routine( void );
//void ConvertNumberToHexString( long number, int *string_ptr, int string_length );

static struct
{
        unsigned int afc_threat         : 3;
        unsigned int afc_control        : 5;
} xdata AFC_data;
struct
{
        unsigned char link_data[15];
        unsigned int data_cnt;
} TxBuf,RxBuf;
unsigned int xdata FreqAddr[10]
        = {0x200, 0x204, 0x208, 0x20C, 0x210, 0x220, 0x224, 0x228, 0x22C, 0x230};
unsigned int xdata SpotBarrageAddr[10]
        = {0x203, 0x207, 0x20B, 0x20F, 0x21D, 0x223, 0x227, 0x22B, 0x22F, 0x23D};
unsigned int xdata RampAddr[10]
        = {0x218, 0x219, 0x21A, 0x21B, 0x21C, 0x238, 0x239, 0x23A, 0x23B, 0x23C};
unsigned int xdata YTFAddr[2] = {0x215, 0x235};
unsigned int xdata AFCAddr[2] = {0x216, 0x236};

unsigned int xdata TxRunning, technique, temp, data_flag, AFCdata, link;
unsigned int xdata threat_value, threat, channel, tech_type,YTF_data, time_count, time_status;
int xdata frequency, freq_data;

void ConfigureTimer( unsigned char timer_id, unsigned char timer_mode )
{
        unsigned char timer_mode_reg;

        timer_mode_reg = TMOD;
        if( timer_id == 0 )
        {
                timer_mode_reg = timer_mode_reg & 0xF0;
                timer_mode_reg |= timer_mode;
                TR0 = 1;
        }
        else
        {
                timer_mode_reg = timer_mode_reg & 0x0F;
```

```
                timer_mode_reg |= (timer_mode << 4);
                TR1 = 1;
                ET1 = 1;
        }
        TMOD = timer_mode_reg;
}

void ConfigureSerialPort( void )
{
        SCON = 0x50;
        PCON = 0x80;
        TL1 = 0xFA;
        TH1 = 0xFA;/* for 9600 baud with 10 MHz clock*/
        ConfigureTimer( 1, 2 );
}

void main( void )
{
        TxRunning = 0;
        TI = 0;
        RI = 0;
        ES = 1;
        EA = 1;
        time_count = 0;
        time_status = 0;
        ConfigureSerialPort( );
        delay();
        while ( link == 1 )
        {
                SBUF = LINK_CON;
                TxRunning = 1;
        }
        TxBuf.link_data[0] = SOF;
        TxBuf.link_data[1] = 'S';
        TxBuf.link_data[2] = 'Y';
        TxBuf.link_data[3] = 'N';
        TxBuf.link_data[4] = 'T';
        TxBuf.link_data[5] = 'H';
        TxBuf.link_data[6] = '  ‘;
        TxBuf.link_data[7] = 'J';
        TxBuf.link_data[8] = 'I';
        TxBuf.link_data[9] = 'G';
```

```
TxBuf.link_data[10] = '\n';
TxBuf.link_data[11] = EOF1;

while (1)
{
        if ( RxBuf.link_data[0] == SOF )
                continue;

        frequency = ConvertHexStringToNumber( &RxBuf.link_data[1], 4 );
        threat = ConvertHexStringToNumber( &RxBuf.link_data[5], 1 );
        channel = ConvertHexStringToNumber( &RxBuf.link_data[6], 1 );
        tech_type = ConvertHexStringToNumber( &RxBuf.link_data[7], 1 );
        technique = ConvertHexStringToNumber( &RxBuf.link_data[8], 1 );
        threat = threat-1;
        generate_rst( );
        RxBuf.link_data[0] = 0;
        if ( threat <= 4 )
                ps_flag = 0;
        else if ( ( threat > 4 ) && ( threat <= 9 ) )
                ps_flag = 1;

        if ( ( threat == 0 ) || ( threat == 5 ) )
                threat_value = 0x0;

        else if ( ( threat == 1 ) || ( threat == 6 ) )
                threat_value = 0x1;

        else if ( ( threat == 2 ) || ( threat == 7 ) )
                threat_value = 0x2;

        else if ( ( threat == 3 ) || ( threat == 8 ) )
                threat_value = 0x3;

        else if ( ( threat == 4 ) || ( threat == 9 ) )
        {
                threat_value = 0x4;

                if( frequency < 26000 )
                        frequency = frequency - 8800;
                else if( frequency < 32000 )
                        frequency = frequency - 17000;
                else
```

```
                frequency = frequency - 22000;
}
freq_data = ConvFreqToDTOWord( frequency);
threat_enable( FreqAddr[threat], threat_value );

AFC_data.afc_threat = threat_value;
AFC_data.afc_control = 0xf;
AFCdata = ( ( AFC_data.afc_control << 3 ) | (AFC_data.afc_threat & 0x7 ) );
if (tech_type == 1)
{
        afclt_flag = 1;
        delay();
        generate_addr_data ( FreqAddr[threat], freq_data );
        delay();
}
else if (tech_type == 2)
{
        afclt_flag = 0;
        delay();
        generate_addr_data ( FreqAddr[threat], freq_data );
        delay();
        generate_addr_data ( RampAddr[threat],RAMP_VALUE);
        delay();
        if (technique == 1)
        {
                generate_addr_data ( SpotBarrageAddr[threat], 0x10 );
                delay();
   }
        else if (technique == 2)
        {
                generate_addr_data ( SpotBarrageAddr[threat], 0x14 );
                delay();
        }
        else if (technique == 3)
        {
                generate_addr_data ( SpotBarrageAddr[threat], 0x18 );
                delay();
        }
        else if (technique == 4)
        {
                generate_addr_data ( SpotBarrageAddr[threat], 0x1C );
                delay();
```

```
        }
}
else if (tech_type == 3)
{
        afclt_flag = 0;
        delay();
        generate_addr_data ( FreqAddr[threat], freq_data );
        delay();
        generate_addr_data ( RampAddr[threat],RAMP_VALUE);
        delay();
        if (technique == 1)
        {
                generate_addr_data ( SpotBarrageAddr[threat],0x00 );
                delay();
        }
        else if (technique == 2)
        {
                generate_addr_data ( SpotBarrageAddr[threat],0x04 );
                delay();
        }
        else if (technique == 3)
        {
                generate_addr_data ( SpotBarrageAddr[threat],0x08 );
                delay();
        }
        else if (technique == 4)
        {
                generate_addr_data ( SpotBarrageAddr[threat],0x0C );
                delay();
        }
}
else if (tech_type == 4)
{
        afclt_flag = 1;
        delay();
        generate_addr_data ( FreqAddr[threat], freq_data );
        delay();
        YTF_data = ConvFreqToYTFWord( frequency );
        generate_addr_data ( YTFAddr[ps_flag], YTF_data );
        delay();
        generate_addr_data ( AFCAddr[ps_flag], AFCdata );
        delay();
```

```
}
else if (tech_type == 5)
{
        TL0 = 0;
        TH0 = 0xD5;
        ConfigureTimer( 0, 0 );
        ET0 = 1;
        while( RxBuf.link_data[0] != SOF )
        {
                if( (time_count <= 50) && (time_status == 1) )
                {
                        afclt_flag = 1;
                        time_status = 0;
                        delay();
                        generate_addr_data ( FreqAddr[threat], freq_data );
                        delay();
                        YTF_data = ConvFreqToYTFWord( frequency );
                        generate_addr_data ( YTFAddr[ps_flag], YTF_data );
                        delay();
                        generate_addr_data ( AFCAddr[ps_flag], AFCdata );
                        delay();
                }
                else if( (time_count > 50) && (time_status == 1) )
                {
                        afclt_flag = 0;
                        delay();
                        generate_addr_data ( FreqAddr[threat], freq_data );
                        delay();
                        generate_addr_data                                      (
RampAddr[threat],RAMP_VALUE);
                        delay();
                        if (technique == 1)
                        {
                        generate_addr_data ( SpotBarrageAddr[threat],0x10 );
                                delay();
                   }
                        else if (technique == 2)
                        {
                        generate_addr_data ( SpotBarrageAddr[threat],0x14 );
                                delay();
                        }
                        else if (technique == 3)
```

```
                                        {
                                        generate_addr_data ( SpotBarrageAddr[threat],0x18 );
                                                delay();
                                        }
                                        else if (technique == 4)
                                        {
                                        generate_addr_data(SpotBarrageAddr[threat],0x1C);
                                                delay();
                                        }

                                }
                        }
                }
        }
}

int ConvFreqToYTFWord( int freq )
{
        int YTFtuningword;

        YTFtuningword = ((freq-7500)*4095L)/10500;
        YTFtuningword = ~YTFtuningword;
        YTFtuningword = 0x0fff & YTFtuningword;
        return (YTFtuningword);
}

int ConvFreqToDTOWord( int freq )
{
        unsigned int Dto_word;

        if(freq < 12000)
                Dto_word = 15 + ((10*(freq-7500))/12);
        else
        {
                Dto_word = 15 + ((10*(freq-12000))/15);
                Dto_word = 0x1000 | Dto_word;
        }
        return (Dto_word);
}

void generate_tgwr( void )
{
```

```
        tgwr_flag = 1;
        delay();
        tgwr_flag = 0;
}

void generate_rst( void )
{
        reset_flag = 1;
        delay();
        reset_flag = 0;
}

void delay( void )
{
        int i;
        for( i = 0; i < 5000; i++ );
}

void threat_enable( unsigned int addr, unsigned int threat_data )
{
        threat_flag = 1;
        out_addr_data( addr, threat_data );
        delay();
        threat_flag = 0;
}

void generate_addr_data (unsigned int address, unsigned int out_data )
{
        addr_flag = 1;
        data_low_flag = 1;
        out_addr_data( address, (out_data & 0xff) );
        delay();
        addr_flag = 0;
        data_low_flag = 0;
        data_high_flag = 1;
        out_addr_data( address, ( (out_data >> 8) & 0xff) );
        delay();
        data_high_flag = 0;
        generate_tgwr( );
}

void out_addr_data( unsigned int addr, unsigned int value )
```

```
{
        unsigned int xdata *address;

        address = addr;
        *address = value;
}

void SerialInterruptHandler(void) interrupt 4
{
        if( TI )
        {
                TI = 0;
                if( TxBuf.link_data[TxBuf.data_cnt] != EOF )
                {
                        SBUF = TxBuf.link_data[TxBuf.data_cnt];
                        TxBuf.data_cnt++;

                }
                else
                {
                        SBUF = TxBuf.link_data[TxBuf.data_cnt];
                        TxBuf.data_cnt = 0;
                        TxRunning = 0;
                }
        }
        if( RI )
        {
                RI = 0;
                RxBuf.link_data[RxBuf.data_cnt] = SBUF;
                if( ( RxBuf.link_data[RxBuf.data_cnt]== SOF ) || (RxBuf.data_cnt > 0) )
                {
                        if( RxBuf.link_data[RxBuf.data_cnt] == EOF1 )
                                RxBuf.data_cnt=0;
                        else if ( RxBuf.link_data[RxBuf.data_cnt]== SOF )
                                RxBuf.data_cnt++;
                        else
                                RxBuf.data_cnt++;
                }
                else if( RxBuf.link_data[RxBuf.data_cnt] == LINK_REQ )
                        link = 1;
        }
}
```

```
void Timer1InterruptHandler(void) interrupt 0
{
	TL0 = 0;
	TH0 = 0xD5;
	ET0 = 0;

	time_count++;
	if( time_count == 1 || ( time_count == 51 ) )
		time_status = 1;
	if( time_count >= 100 )
	{
		time_count = 0;
		time_status = 0;
	}
	ET0 = 1;
}

unsigned int ConvertHexStringToNumber( unsigned char *temp_buf, int string_length )
{
	unsigned char temp_idx;
	unsigned int number;

	number = 0;
	do
	{
		if( (temp_buf[temp_idx] == '\0') || (temp_buf[temp_idx] == ' ')
			|| (temp_buf[temp_idx] == '\n') || (temp_buf[temp_idx] == '\r') )
			break;
		for ( temp_idx = string_length-1; temp_idx <= 0; temp_idx--)
		{
			number = number << 4;
			if( (temp_buf[temp_idx] >= '0') && (temp_buf[temp_idx] <= '9') )
				number = number + (temp_buf[temp_idx]-'0');
			else if( (temp_buf[temp_idx] >= 'a') && (temp_buf[temp_idx] <= 'f') )
				number = number + 0xA + (temp_buf[temp_idx]-'a');
			else if( (temp_buf[temp_idx] >= 'A') && (temp_buf[temp_idx] <= 'F') )
				number = number + 0xA + (temp_buf[temp_idx]-'A');
			else
			{
			number = number >> 4;
			return -1;
```

```
                        }
                    }
        } while( 1 );
        return number;
}
```

PROGRAM 5: TO BE WRITTEN IN VHDL FOR FPGA

PROGRAM FOR DECODER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity decoder is
   Port ( clk : in  STD_LOGIC;
          en : in  STD_LOGIC;
          in_data1 : in  STD_LOGIC_VECTOR (2 downto 0);
          out_data1 : out  STD_LOGIC_VECTOR (4 downto 0));
end decoder;

architecture Behavioral of decoder is

begin

process (clk)
begin
if ( clk'event and clk ='1') then
     if en = '1' then
       case in_data1 is
          when "000" => out_data1 <= "00001";
          when "001" => out_data1 <= "00010";
          when "010" => out_data1 <= "00100";
          when "011" => out_data1 <= "01000";
          when "100" => out_data1 <= "10000";

when others => out_data1 <= "00000";
        end case;
     end if;
  end if;
end process;
end Behavioral;
```

PROGRAM FOR LATCHES

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity latch is
   Port ( clk : in  STD_LOGIC;
                                    en : in  STD_LOGIC;
          in_data : in  STD_LOGIC_VECTOR (7 downto 0);
                                out_data : out  STD_LOGIC_VECTOR (7 downto 0));
end latch;

architecture Behavioral of latch is

begin
process (clk)
begin

if clk='1' and clk'event then
          if en='1' then
                    out_data <= in_data;
          end if;
end if;
end process;

end Behavioral;
```

MAIN PROGRAM FOR FPGA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity main is
   Port (  --rd : in  STD_LOGIC;
                                  --wr : in  STD_LOGIC;
                              -- ale : in  STD_LOGIC;
           addr_data : in  STD_LOGIC_VECTOR (7 downto 0);
                              addr : in  STD_LOGIC_VECTOR (7 downto 0);
                              control : in  STD_LOGIC_VECTOR (7 downto 0);

                                  fpgarst : out  STD_LOGIC;
                                  afclt : out  STD_LOGIC;
                                  syn_ps : out  STD_LOGIC;
                                  tgwr : out  STD_LOGIC;
          address : out  STD_LOGIC_VECTOR (15 downto 0);
                              data : out  STD_LOGIC_VECTOR (15 downto 0);
                              tsel : out  STD_LOGIC_VECTOR (4 downto 0));
end main;

architecture Behavioral of main is

component latch is
   Port ( clk : in  STD_LOGIC;
                                  en : in  STD_LOGIC;
          in_data : in  STD_LOGIC_VECTOR (7 downto 0);
                              out_data : out  STD_LOGIC_VECTOR (7 downto 0));
end component;

component decoder is
   Port ( clk : in STD_LOGIC;
                                  en : in  STD_LOGIC;
          in_data1 : in  STD_LOGIC_VECTOR (2 downto 0);
                              out_data1 : out  STD_LOGIC_VECTOR (4 downto 0));
end component;

signal en1, en2, en3, en4, en5, en6,en7, fpgarst1 : STD_LOGIC;
signal lat_addr : STD_LOGIC_VECTOR (15 downto 0);
signal lat_tsel : STD_LOGIC_VECTOR (4 downto 0);
```

```
signal lat_data : STD_LOGIC_VECTOR (15 downto 0);

begin

en1 <= control(0) and (not control(3));
en2 <= control(7) and (not control(3));
en3 <= control(1) and (not control(3));
en4 <= control(2) and (not control(3));
en5 <= control(4);
tgwr <= en5;
fpgarst1 <= control(6);
fpgarst <= control(6);
en6 <= '1';
syn_ps <= '0';
en7 <= control(3);

process(en5)
        begin
                if en5='1' and en5'event then
                address(15 downto 0) <= lat_addr(15 downto 0);
                tsel(4 downto 0) <= lat_tsel(4 downto 0);
                data(15 downto 0) <= lat_data(15 downto 0);
                afclt <= control(5);
        end if;
end process ;

process(fpgarst1)
begin
        if fpgarst1 ='0' and fpgarst1'event then
        data(15 downto 0) <= "ZZZZZZZZZZZZZZZZ";
        address(15 downto 0) <= "ZZZZZZZZZZZZZZZZ";
        tsel(4 downto 0) <= "ZZZZZ";
        end if;
End process;

latch1: latch port map (clk => en1,
                                en => en6, in_data => addr,
                        out_data(7 downto 0) => lat_addr(7 downto 0));

latch2: latch port map (clk => en2,
                                en => en6, in_data  => addr,
                        out_data(7 downto 0)=> lat_addr(15 downto 8));
```

```
latch3: latch port map (clk => en3,
                        en => en6, in_data => addr_data,
                    out_data(7 downto 0) => lat_data(7 downto 0));

latch4 : latch port map (clk => en4,
                        en => en6, in_data => addr_data,
                    out_data(7 downto 0) => lat_data(15 downto 8));

decoder1 : decoder port map ( clk => en7,
                        en => en6,
        in_data1 => addr_data(2 downto 0),
                    out_data1 => lat_tsel);

end Behavioral;
```