Application Note

AN_260

IR Remote Control Implementation by Java D2XX

Version 1.0

Issue Date: 2013-09-25

This application note illustrates how NEC IR (38 KHz), RC 5, and RC 6 waveforms are generated by an Android device using Java D2XX with an FT-X chip working in Async bit-bang mode with minimum hardware cost. Also illustrates using Sync bit-bang mode to perform continuous reading.

Use of FTDI devices in life support and/or safety applications is entirely at the user's risk, and the user agrees to defend, indemnify and hold FTDI harmless from any and all damages, claims, suits or expense resulting from such use.

Future Technology Devices International Limited (FTDI)
Unit 1, 2 Seaward Place, Glasgow G41 1HH, United Kingdom
Tel.: +44 (0) 141 429 2777   Fax: + 44 (0) 141 429 2758
Web Site: http://ftdichip.com

Table of Contents

1 Introduction

This application note illustrates how NEC IR (38 KHz), RC 5, and RC 6 waveforms are generated by an Android device using Java D2XX with an FT-X chip working in Async bit-bang mode with minimum hardware cost. Also illustrates using Sync bit-bang mode to perform continuous reading.

1.1 Overview

This document is designed for engineers who want to design an IR module with simple SetBitMode, Read and Write commands for an Android system.

1.2 Scope

NEC, Philip RC-5, and Philip RC-6 IR waveforms are described in following sections.

2 Infra Red (IR) Data Protocols

Infra red data is typically serial data. It is encoded to different protocols that different vendors support. As an example such encoding is what allows a TV remote control to work with one TV but not another.

2.1 NEC IR Protocol

NEC IR features are listed below

- 8 bit address and 8 bit command
- Address and command are transmitted twice
- Pulse distance modulation
- Carrier frequency is 38KHz
- Bit time is 1.125 ms (logical 0) or 2.25 ms (logical 1)

NEC IR protocol is listed as following

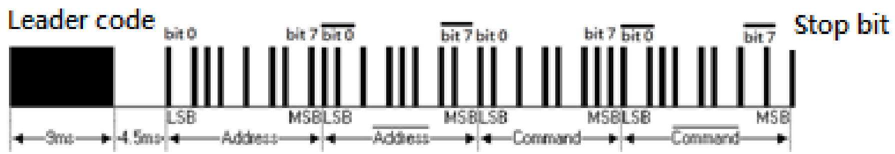

Leader code
bit 0
bit 7
$\overline{\text{bit 0}}$
$\overline{\text{bit 7}}$
bit 0
bit 7
$\overline{\text{bit 0}}$
$\overline{\text{bit 7}}$
Stop bit
LSB
MSB
LSB
MSB
LSB
MSB
LSB
MSB
9ms
4.5ms
Address
$\overline{\text{Address}}$
Command
$\overline{\text{Command}}$

Figure 2.1 NEC IR Protocol

NEC IR modulation

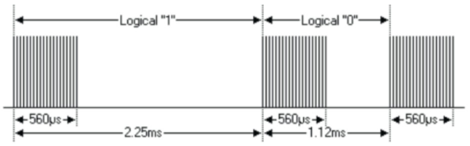


Figure 2.2 NEC IR Modula

NEC IR carrier

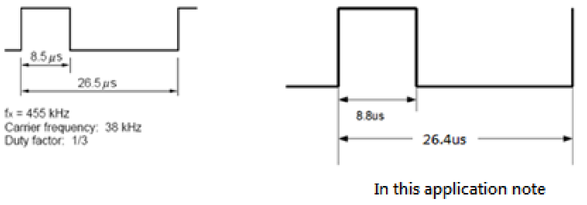
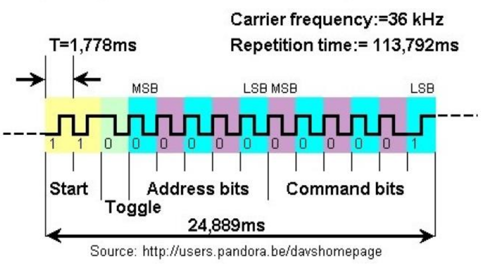


Figure 2.3 NEC IR Carrier

2.2 Philips RC-5 IR Protocol

Philips RC-5 IR features are listed below

- 5 bit address and 6 bit command
- All bits have an equal length of 1.778ms
- Bi-phase modulation (also called Manchester coding)
- Carrier frequency is 36KHz
- The duty cycle of carrier frequency is 25% to 33%



Source: http://users.pandora.be/davshomepage

Figure 2.4 Philips RC-5 IR Protocol

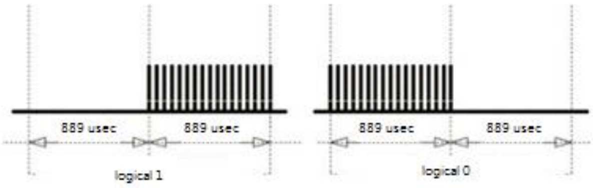


Figure 2.5 Philips RC-5 IR Modula

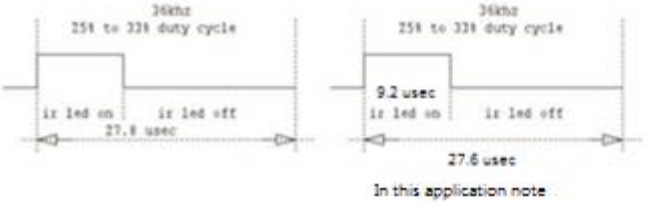


Figure 2.6 Philips RC-5 IR Carrier

2.3 Philips RC-6 IR Protocol

RC-6 signals are modulated on a 36 kHz carrier the same as RC-5. The duty cycle of this carrier has to be between 25% and 50%.

3 Hardware Requirement

The following hardware is required to test the IR transmitter on an Android platform.

An Android system with an OTG USB connector (Google Nexus 7 in this application note)

An FTDI UMFT234XD module

A NPN transistor (BC817 in this application note)

An IR LED

1KΩ and 100Ω resistors

3.1 Async BitBang Mode of FTDI Devices

Async BitBang Mode is able to set each pin independently as an input or an output. The data rate is controlled by the baud rate generator. New data is written when the baud rate clock ticks. If no new data is written to the device, the pins will hold the last written value. This application uses only the TXD pin to generate IR LED signal(eg. 4 bytes of data are needed to send a data stream 0101.).
The clock for the BitBang mode is actually 16 times the baudrate. A value of 9600 baud would transfer the data at (9600 x 16) = 153600 bytes per second or once every 6.5 uS.

Set the baud rate to 7102 to have 8.8 usec duty on cycle in this application note.

3.2 Sync BitBang Mode of FTDI Devices

Synchronous Bit Bang mode, data will only be sent out if there is space in the device for data to be read from the pins. This Synchronous Bit Bang mode will read the data bus pins first, before it sends out the byte that has just been transmitted. It is therefore 1 byte behind the output and so to read the inputs for the byte that you have just sent, another byte must be sent." (AN_232R-01) Sync Bit Bang mode is used for IR signals reading in this application.

Set the baud rate to 625. The sample reate will be 1/(625*16) = 100us. The program will read IR signals every 100 us. It is fast enough to sample NEC IR signals since stop bit is 560us, Logic "0" is 1.12ms, and Logic "1" is 2.25ms.

4 Example Circuit

4.1 Typical UART to IR block diagram

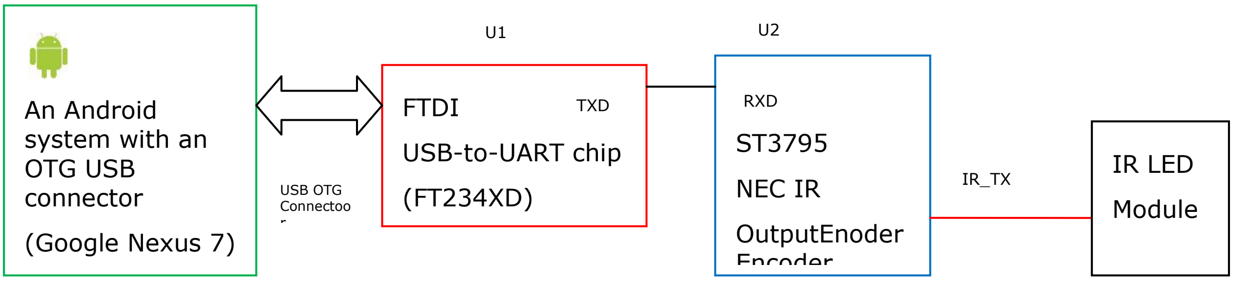


Figure 4.1 Typical USB-to-IR block diagram

A typical IR design will take UART information from an MCU (USB/UART bridge in this diagram) and buffer this data through a dedicated IR transmitter.

4.2 A minimum hardware cost IR block diagram

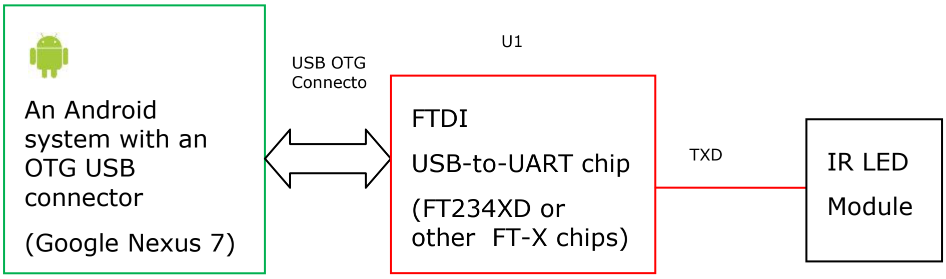

An Android system with an OTG USB connector
(Google Nexus 7)
USB OTG Connecto
U1
FTDI
USB-to-UART chip
(FT234XD or other FT-X chips)
TXD
IR LED
Module

Figure 4.2 A minimum USB-to-IR block diagram

Using an FTDI chip in bitbang mode whereby a data stream is bit banged out on a pin to the IR module removes the dedicated IR encoder. Encoding is done on the host tablet and bit banged on the FT234XD.

4.3 Schematic

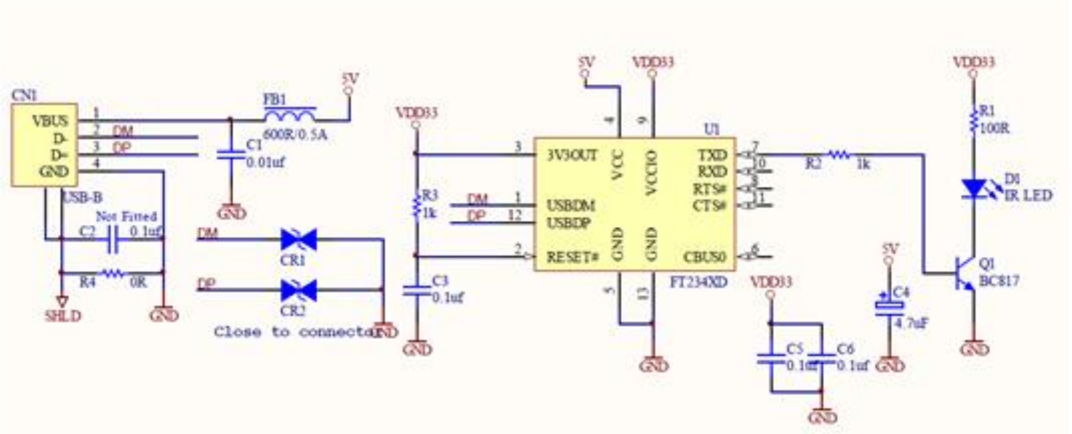
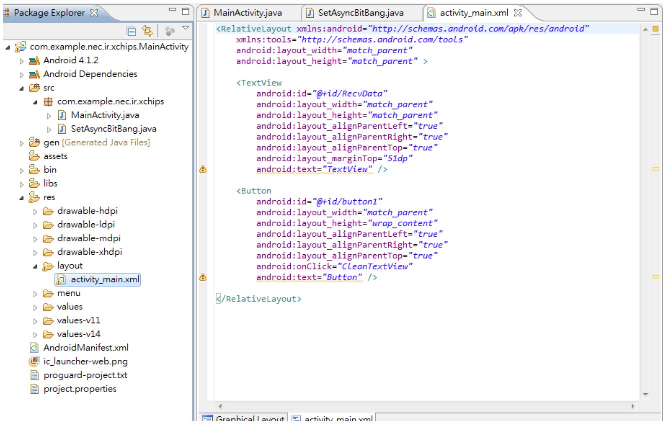


Figure 4.3 Schematic of IR LED module

5 Example Program(writing)

MainActivity.java is the main program and SetAsyncBitBang.java does the mode setting to async bitbang mode.

The button1 click event will call CleanTextView function.



```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:id="@+id/RecvData"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="51dp"
        android:text="TextView" />

    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:onClick="CleanTextView"
        android:text="Button" />

</RelativeLayout>
```

Figure 5.1 Java D2XX code overview

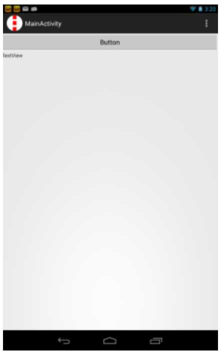


Figure 5.2 Java D2XX code Running Result

5.1 MainActivity.java

```java
package com.example.nec.ir.xchips;

import android.os.Bundle;
import android.app.Activity;
import android.content.pm.ActivityInfo;
import android.text.method.ScrollingMovementMethod;
import android.view.Menu;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity {

        private SetAsyncBitBang MyAsyncBitBang = null;
        public byte[] NECIRAddress = new byte[8];
        public byte[] NECIRCommand = new byte[8];

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

                setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
                TextView MyMessage = (TextView) findViewById(R.id.RecvData);
                MyMessage.setMovementMethod(ScrollingMovementMethod.getInstance());
                IRSetBitMode();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }

        public void CleanTextView(View view) {

                TextView GetMyData = (TextView) findViewById(R.id.RecvData);
                GetMyData.setText("");
                NECIRAddress[0]=0;
                NECIRAddress[1]=0;
                NECIRAddress[2]=0;
                NECIRAddress[3]=0;

                NECIRAddress[4]=0;
                NECIRAddress[5]=0;
                NECIRAddress[6]=0;
                NECIRAddress[7]=0;


                NECIRCommand[0]=1;
                NECIRCommand[1]=1;
                NECIRCommand[2]=0;
                NECIRCommand[3]=1;

                NECIRCommand[4]=1;
                NECIRCommand[5]=0;
                NECIRCommand[6]=1;
                NECIRCommand[7]=0;


                MyAsyncBitBang.Gen38KHzIR(NECIRAddress, NECIRCommand);
        }

        public void onMsgShow(final String recvdata){
                runOnUiThread(new Runnable() {
                        public void run() {
                                TextView showstr = (TextView) findViewById(R.id.RecvData);
                                showstr.append(recvdata);
                        }
                });
        }

        private void IRSetBitMode() {

                MyAsyncBitBang= new SetAsyncBitBang(this);
        }
}
```

5.2 SetAsyncBitBang.java

```
package com.example.nec.ir.xchips;

import android.app.Activity;
import android.content.IntentFilter;
import android.hardware.usb.UsbManager;

import com.ftdi.j2xx.D2xxManager;
import com.ftdi.j2xx.FT_Device;

public class SetAsyncBitBang {

        private Activity mContext;
        private D2xxManager ftD2xx = null;
        private FT_Device ftDevice = null;
        private byte[] DataOutBuffer = new byte[7692];
        public static final int NECIRHeaderLen= 341;
        private byte[] NECIRLeaderCode = new byte[3*NECIRHeaderLen];             // 9ms/26.4us = 341

        public static final int NECIRSpaceLen= 170;
        private byte[] NECIRSpace = new byte[3*NECIRSpaceLen];                            //
4.5m/26.4us = 170

        public static final int NECIROneLen= 85;
        private byte[] NECIROne = new byte[3*NECIROneLen];                                        //
2.25m/26.4us = 85

        public static final int NECIRZeroLen= 42;
        private byte[] NECIRZero = new byte[3*NECIRZeroLen];                              //
1.12m/26.4us = 42


        public static final int AddIdx =3*(NECIRHeaderLen+NECIRSpaceLen);         // 1533

        public static final int CmdIdx =3*(NECIRHeaderLen+NECIRSpaceLen+8*(NECIROneLen+NECIRZeroLen));
        // the index to point the command length 4581

        public static final int NECIRStopLen= 21;
        private byte[] NECIRStopBit = new byte[3*NECIRStopLen];                                    //
560u/26.4us = 21
        public static final int StopIdx
=3*(NECIRHeaderLen+NECIRSpaceLen+2*8*(NECIROneLen+NECIRZeroLen));
        //7629

public SetAsyncBitBang (Activity activity) {
        mContext = activity;
        try{
                ftD2xx = D2xxManager.getInstance(mContext);
        }catch (D2xxManager.D2xxException ex)
        {
                LogInfo(ex.getMessage());
        }
        IntentFilter filter = new IntentFilter();
    filter.addAction(UsbManager.ACTION_USB_DEVICE_DETACHED);
}

public void LogInfo(String val) {
        ((MainActivity)mContext).onMsgShow(val+"\n");
}

public boolean Gen38KHzIR(byte[] nECIRAddress, byte[] nECIRCommand) {
        int i;
        int j;
        int devCount = ftD2xx.createDeviceInfoList(mContext);
        int AddOffset=0;         //The offset of Address
        int CmdOffset=0;         //The offset of Command


        if(devCount > 0)
        {
                D2xxManager.FtDeviceInfoListNode[] deviceList = new
D2xxManager.FtDeviceInfoListNode[devCount];
                ftD2xx.getDeviceInfoList(devCount, deviceList);

                switch (deviceList[0].type) {
                case D2xxManager.FT_DEVICE_232B:
                        LogInfo("Device Name : FT232B device");
                        break;

                case D2xxManager.FT_DEVICE_8U232AM:
                        LogInfo("Device Name : FT8U232AM device");
                        break;

                case D2xxManager.FT_DEVICE_UNKNOWN:
                        LogInfo("Device Name : Unknown device");
                        break;
                case D2xxManager.FT_DEVICE_2232:
                        LogInfo("Device Name : FT2232 device");
                        break;

                case D2xxManager.FT_DEVICE_232R:
                        LogInfo("Device Name : FT232R device");
                        break;

                case D2xxManager.FT_DEVICE_2232H:
                        LogInfo("Device Name : FT2232H device");
                        break;

                case D2xxManager.FT_DEVICE_4232H:
                        LogInfo("Device Name : FT4232H device");
                        break;

                case D2xxManager.FT_DEVICE_232H:
                        LogInfo("Device Name : FT232H device");
                        break;
                case D2xxManager.FT_DEVICE_X_SERIES:
                        LogInfo("Device Name : FTDI X_SERIES");
                        break;
                default:
                        LogInfo("Device Name : FT232B device");
                        break;
                }

                ftDevice = ftD2xx.openByIndex(mContext, 0);
                if(null == ftDevice) {
                LogInfo("ftDevice = null");
                return false;
        }

                // ******************
                // configure our port , Set to ASYNC BIT MODE
                if (!ftDevice.setBitMode((byte) 0xFF, D2xxManager.FT_BITMODE_ASYNC_BITBANG))
                        LogInfo("Set to ASYNC bitbang failed");

                // *************** Configure Baud rate
                //** Reference AN232BM-01 to set the baud rate
                //** To have 8.8us baud rate in my case
                //** 9600 x 16 = 153600
                //** 1/153600 = 6.5us
                //** So 1/113636.36 = 8.8us
                //** 113636.36 / 16 = 7102.27
                if(!ftDevice.setBaudRate(7102))
                        LogInfo("Set bit rate failed");  //8.8us
                //******************

                //****** Header *******
                for (i=0;i<NECIRHeaderLen;i++)
                {
                        NECIRLeaderCode[3*i]=  (byte) 0xFF;
                        NECIRLeaderCode[3*i+1]=  (byte) 0x0;
                        NECIRLeaderCode[3*i+2]=  (byte) 0x0;
                }

                //****** Space ***************
                for (i=0;i<NECIRSpaceLen;i++)
                {
                        NECIRSpace[3*i]= NECIRSpace[3*i+1] =NECIRSpace[3*i+2]=0;
                }

                //********NECIROne********
                for (i=0;i<21;i++)
                {
                        NECIROne[3*i]=  (byte) 0xFF;
                        NECIROne[3*i+1]=  (byte) 0x0;
                        NECIROne[3*i+2]=  (byte) 0x0;
                }
```

```
for (i=22;i< NECIROneLen; i++)
{
        NECIROne[3*i]=NECIROne[3*i+1]=NECIROne[3*i+2]=0;
}

//********NECIRZero********
for (i=0;i<21;i++)
{
        NECIRZero[3*i]=  (byte) 0xFF;
        NECIRZero[3*i+1]=  (byte) 0x0;
        NECIRZero[3*i+2]=  (byte) 0x0;
}
for (i=22;i< NECIRZeroLen; i++)
{
        NECIRZero[3*i]=NECIRZero[3*i+1]=NECIRZero[3*i+2]=0;
}

//********NECIR Stop bit********
for (i=0;i<21;i++)
{
        NECIRStopBit[3*i]=  (byte) 0xFF;
        NECIRStopBit[3*i+1]=  (byte) 0x0;
        NECIRStopBit[3*i+2]=  (byte) 0x0;
}


// *** fill up the IR Leader Code
for (i=0;i<3*NECIRHeaderLen;i++)
{
        DataOutBuffer[i]= NECIRLeaderCode[i];
}

// *** fill up the NEC IR Space
for (i=0;i<3*NECIRSpaceLen;i++)
{
        DataOutBuffer[(3*NECIRHeaderLen)+i]=  NECIRSpace[i];

}


//**** NEC IR Address *****

for (j=0; j<8; j++) {                   // 8 bit of NECIRAddress; put Address

        if (nECIRAddress[j]==0)  // nECIRAddress[j]==0
        {
                for (i=0;i<NECIRZeroLen*3; i++) {
                        DataOutBuffer[AddIdx+AddOffset]=NECIRZero[i];
                        AddOffset++;
                }
        }
        else
        {
                for (i=0;i<NECIROneLen*3; i++) {
                        DataOutBuffer[AddIdx+AddOffset]=NECIROne[i];
                        AddOffset++;
                }
        }
}

// **** NEC IR Address Bar

for (j=0; j<8; j++) {                   // 8 bit of NECIRAddress; put Address Bar
        if (nECIRAddress[j]!=0)  // nECIRAddress[j]!=0 Address Bar
        {
                for (i=0;i<NECIRZeroLen*3; i++) {
                        DataOutBuffer[AddIdx+AddOffset]=NECIRZero[i];
                        AddOffset++;
                }
        } else {
                for (i=0;i<NECIROneLen*3; i++) {
```

```
                                        DataOutBuffer[AddIdx+AddOffset]=NECIROne[i];
                                        AddOffset++;
                                }
                        }
                }

                //**** End of NEC IR Address  *****

                CmdOffset=0;
                //**** NEC IR Command *****

                for (j=0; j<8; j++) {                   // 8 bit of NECIRAddress; put Address

                        if (nECIRCommand[j]==0)  // nECIRAddress[j]==0
                        {
                                for (i=0;i<NECIRZeroLen*3; i++) {
                                        DataOutBuffer[CmdIdx+CmdOffset]=NECIRZero[i];
                                        CmdOffset++;
                                }
                        }
                        else
                        {
                                for (i=0;i<NECIROneLen*3; i++) {
                                        DataOutBuffer[CmdIdx+CmdOffset]=NECIROne[i];
                                        CmdOffset++;
                                }
                        }
                }

                // **** NEC IR Command Bar

                for (j=0; j<8; j++) {                   // 8 bit of NECIRAddress; put Address Bar
                        if (nECIRCommand[j]!=0)  // nECIRAddress[j]!=0 Address Bar
                        {
                                for (i=0;i<NECIRZeroLen*3; i++) {
                                        DataOutBuffer[CmdIdx+CmdOffset]=NECIRZero[i];
                                        CmdOffset++;
                                }
                        } else {
                                for (i=0;i<NECIROneLen*3; i++) {
                                        DataOutBuffer[CmdIdx+CmdOffset]=NECIROne[i];
                                        CmdOffset++;
                                }
                        }
                }

                //**** End of NEC IR Command  *****

                //********** NEC IR Stop bit  *******
                for (i=0; i < 3*NECIRStopLen; i++)
                        DataOutBuffer[StopIdx+i]= NECIRStopBit[i];
          for (i=0; i<1; i++)
          {
                  ftDevice.write(DataOutBuffer, 7692);
          }
                ftDevice.close();
                return true;
        }
        return false;
}

}
```

6 Example application (reading)

To test the IR reader a sample waveform was generated from a PC application driving the IR output (section 6.1). This is then used to drive the IR reader being run on the Android platform.

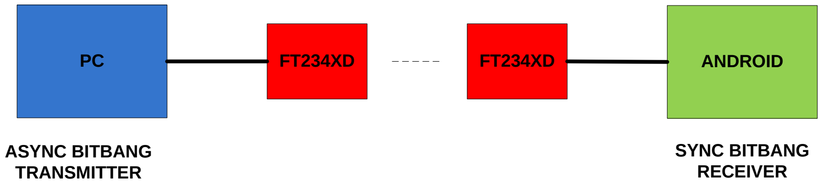


Figure 6.1 IR Reader Test Setup

6.1 Sample application to drive the IR reader

The following Visual C++ program is able to generate different Baud Rates
MyAsyncBitBang.cpp A Visual C++ application to test the carrier waveform

```
// MyAsyncBitBang.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include <windows.h>
#include "FTD2XX.h"
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#define REPEAT 18020

FT_STATUS ftStatus;

DWORD dwNumBytesToSend = 0;     //Index of output buffer
DWORD dwNumBytesSent = 0, dwNumBytesRead = 0, dwNumInputBuffer = 0;

BYTE ByteDataRead;

WORD i=0;
int j;
BYTE DataOutBuffer[REPEAT];

BYTE InputBuffer[1024];
// int BAUDRATE=7102;           // 8.8 usec
int BAUDRATE=6793;              //  9.2u usec

BOOL Bitbang_Initial(FT_HANDLE ftHandle)
{

        ftStatus |= FT_SetLatencyTimer(ftHandle, 2);            //Set the latency timer
        ftStatus |= FT_SetBitMode(ftHandle, 0x0, 0x00);         //Reset controller
        ftStatus |= FT_SetBitMode(ftHandle, 0xff, 0x01);        //Enable Async Bitbang mode

        ftStatus |= FT_SetBaudRate(ftHandle, BAUDRATE);
        if (ftStatus != FT_OK)
        {
                printf("fail on initialize Async Bit-bang mode ! \n");
                return false;
        }
        printf("Async Bitbang mode initial successful\n");
        return true;
}

int _tmain(int argc, _TCHAR* argv[])
{
        int i,j;
  FT_HANDLE ftdiHandle;
  ftStatus = FT_Open(0,&ftdiHandle);

  if(Bitbang_Initial(ftdiHandle) == TRUE)
  {
        for (j=0;j<6000; j++)
        {
                        // Out 27.6 usec with 9.2usec duty cycle
                        DataOutBuffer[3*j] = 0xff; // out 9.2usec high

                        DataOutBuffer[3*j+1] = 0x00; // output (27.6-9.2) usec low
                        DataOutBuffer[3*j+2] = 0x00; // output (27.6-9.2) usec low
        }

        for (j=0; j<1000; j++)
        {
                ftStatus = FT_Write(ftdiHandle,DataOutBuffer, 18000, &dwNumBytesSent);
        }
 }
  printf("Exit the loop now.\n");
  FT_Close(ftdiHandle);
  printf("Closed the file handle.\n");
  getch();

  return 0;
}
```

6.2 JAVA D2xx Sync BitBang Reader App

An example to read IR signals continually on bit 0 of FT-X chip. Again MainActivity.java is the main application while syncbitbang.java controls the bitbang interface for reading the data.

MainActivity.java

```
package com.example.syncbitbang4ir;

import com.ftdi.j2xx.D2xxManager;
import com.ftdi.j2xx.FT_Device;

import android.app.Activity;
import android.content.IntentFilter;
import android.hardware.usb.UsbManager;

public class SyncBitBang {

        CompareThread compareThread;

        private Activity mContext;
        private D2xxManager ftD2xx = null;
        private FT_Device ftDevice = null;
        public static final int TwoK= 2*1024;

        private byte[] DataOutBuffer = new byte[TwoK];
        private byte[] DataInBuffer = new byte[TwoK];            // get data to this buffer
        public SyncBitBang (Activity activity) {
                mContext = activity;
                try{
                        ftD2xx = D2xxManager.getInstance(mContext);
                }catch (D2xxManager.D2xxException ex)
                {
                        LogInfo(ex.getMessage());
                }
                IntentFilter filter = new IntentFilter();
            filter.addAction(UsbManager.ACTION_USB_DEVICE_DETACHED);
        }

        public void LogInfo(String val) {
                ((MainActivity)mContext).onMsgShow(val+"\n");
        }

        public boolean Gen128KHzIR() {
                int devCount = ftD2xx.createDeviceInfoList(mContext);

                // LogInfo("Inside of Gen128KHzIR function...");
                 if(devCount > 0)
                {

                        D2xxManager.FtDeviceInfoListNode[] deviceList = new
D2xxManager.FtDeviceInfoListNode[devCount];
                        ftD2xx.getDeviceInfoList(devCount, deviceList);

                        //*** send data and loop back****

                        // **************** Configure Baud rate
                        //** Reference AN232BM-01 to set the baud rate
                        //** To have 8.8us baud rate in my case
                        //** 9600 x 16 = 153600
                        //** 1/153600 = 6.5us
                        //** So 8.8us = 1/113636.36
                        //** 100 us = 1/(625*16)
                        // if(!ftDevice.setBaudRate(8000))      //128Khz
                        if(!ftDevice.setBaudRate(625))   //10kHz
                                LogInfo("Set bit rate failed");
                        //******************

                        if(!ftDevice.setLatencyTimer((byte) 2)); //Set Latency time to 2ms
                                LogInfo("Set latency time failed");
                        //******************
                        // Reset
                        if (!ftDevice.setBitMode((byte) 0xFE, D2xxManager.FT_BITMODE_RESET))
                                LogInfo("Reset failed");

                        // Sync Write also read back
                        // configure our port , Set to SYNC BIT MODE
                        if (!ftDevice.setBitMode((byte) 0xFE, D2xxManager.FT_BITMODE_SYNC_BITBANG)) //
bit 0 input, bit 1~7 output
                                LogInfo("Set to SYNC bitbang failed");
                        else
                                LogInfo("SYNC bitbang going...");

                        compareThread = new CompareThread();
                        compareThread.start();
                        return true;
                }
                 return false;
                }

        class CompareThread extends Thread
        {
                public void run()
                {
                        int i;

                        ftDevice.write(DataOutBuffer, 256);     // Sync write 256 bytes
                        for (i=0;i<5000; i++) {

                                ftDevice.write(DataOutBuffer, 256);      // Sync another 256 bytes
                                ftDevice.read(DataInBuffer, 256);        // Sync read 256 bytes
                        }
                        ftDevice.close();
                }
        }

}
```

SyncBitBang.java

```
package com.example.syncbitbang4ir;

import com.ftdi.j2xx.D2xxManager;
import com.ftdi.j2xx.FT_Device;

import android.app.Activity;
import android.content.IntentFilter;
import android.hardware.usb.UsbManager;

public class SyncBitBang {

        CompareThread compareThread;

        private Activity mContext;
        private D2xxManager ftD2xx = null;
        private FT_Device ftDevice = null;
        public static final int TwoK= 2*1024;

        private byte[] DataOutBuffer = new byte[TwoK];
        private byte[] DataInBuffer = new byte[TwoK];           // get data to this buffer
        public SyncBitBang (Activity activity) {
                mContext = activity;
                try{
                        ftD2xx = D2xxManager.getInstance(mContext);
                }catch (D2xxManager.D2xxException ex)
                {
                        LogInfo(ex.getMessage());
                }
                IntentFilter filter = new IntentFilter();
            filter.addAction(UsbManager.ACTION_USB_DEVICE_DETACHED);
        }

        public void LogInfo(String val) {
                ((MainActivity)mContext).onMsgShow(val+"\n");
        }

        public boolean Gen128KHzIR() {
                int devCount = ftD2xx.createDeviceInfoList(mContext);

                // LogInfo("Inside of Gen128KHzIR function...");
                 if(devCount > 0)
                {

                        D2xxManager.FtDeviceInfoListNode[] deviceList = new
D2xxManager.FtDeviceInfoListNode[devCount];
                        ftD2xx.getDeviceInfoList(devCount, deviceList);

                        //*** send data and loop back****

                        // *************** Configure Baud rate
                        //** Reference AN232BM-01 to set the baud rate
                        //** To have 8.8us baud rate in my case
                        //** 9600 x 16 = 153600
                        //** 1/153600 = 6.5us
                        //** So 8.8us = 1/113636.36
                        //** 100 us = 1/(625*16)
                        // if(!ftDevice.setBaudRate(8000))      //128Khz
                        if(!ftDevice.setBaudRate(625))  //10kHz
```

```
                                LogInfo("Set bit rate failed");
                        //******************
                        if(!ftDevice.setLatencyTimer((byte) 2)); //Set Latency time to 2ms
                                LogInfo("Set latency time failed");
                        //******************
                        // Reset
                        if (!ftDevice.setBitMode((byte) 0xFE, D2xxManager.FT_BITMODE_RESET))
                                LogInfo("Reset failed");

                        // Sync Write also read back
                        // configure our port , Set to SYNC BIT MODE
                        if (!ftDevice.setBitMode((byte) 0xFE, D2xxManager.FT_BITMODE_SYNC_BITBANG)) //
bit 0 input, bit 1~7 output
                                LogInfo("Set to SYNC bitbang failed");
                        else
                                LogInfo("SYNC bitbang going...");

                        compareThread = new CompareThread();
                        compareThread.start();
                        return true;
                }
                 return false;
                }
        class CompareThread extends Thread
        {
                public void run()
                {
                        int i;

                        ftDevice.write(DataOutBuffer, 256);     // Sync write 256 bytes

                        for (i=0;i<5000; i++) {

                                ftDevice.write(DataOutBuffer, 256);     // Sync another 256 bytes
                                ftDevice.read(DataInBuffer, 256);       // Sync read 256 bytes
                        }
                        ftDevice.close();
                }
        }

}
```

7 Testing Results

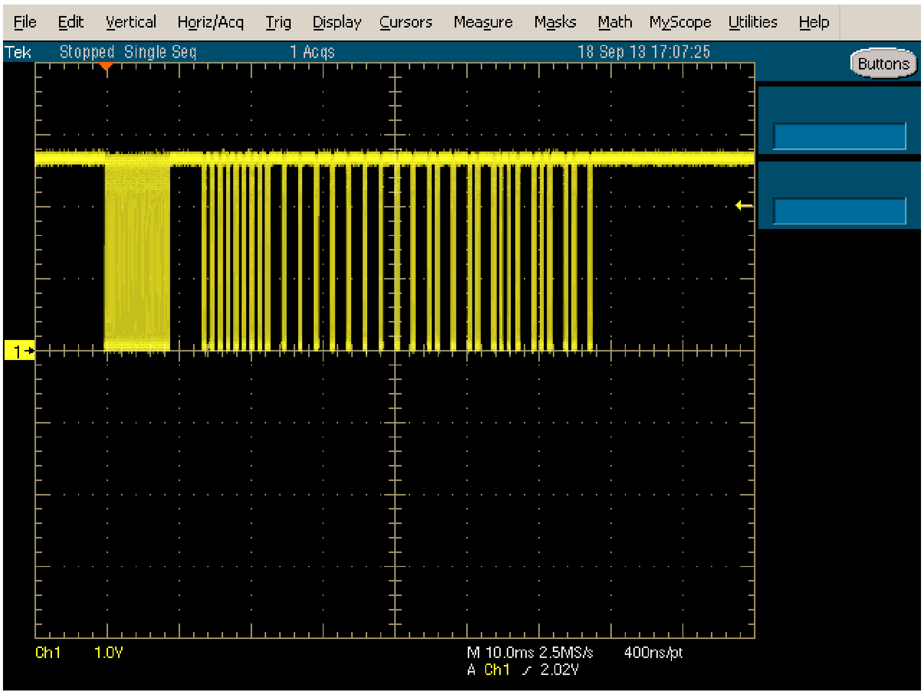
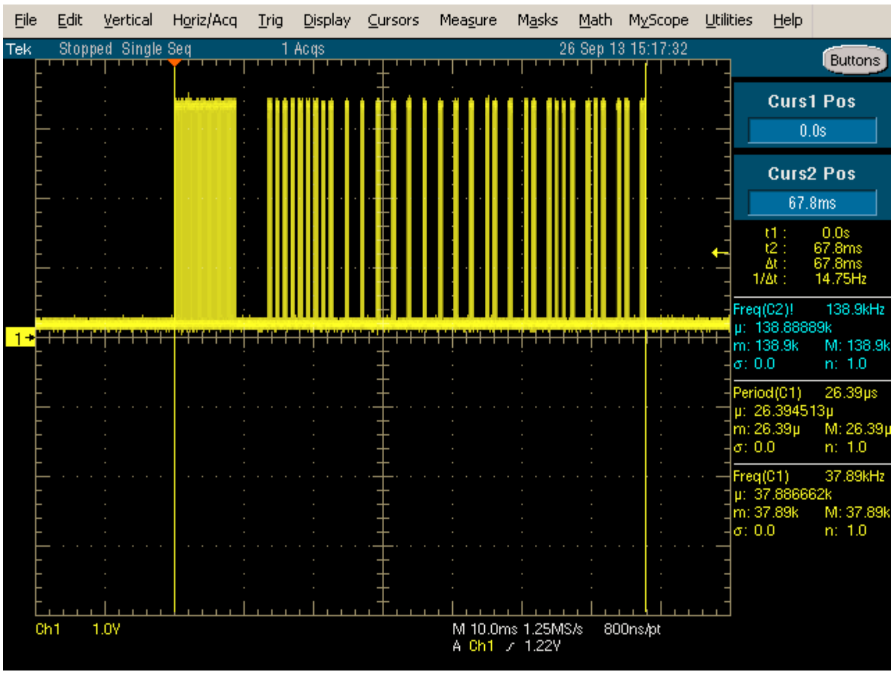

File Edit Vertical Horiz/Acq Trig Display Cursors Measure Masks Math MyScope Utilities Help
Tek Stopped Single Seq 1 Acqs 18 Sep 13 17:07:25
Buttons
1
Ch1 1.0V
M 10.0ms 2.5MS/s 400ns/pt
A Ch1 ⁄ 2.02V

Figure 7.1 Measure the IR LED to GND

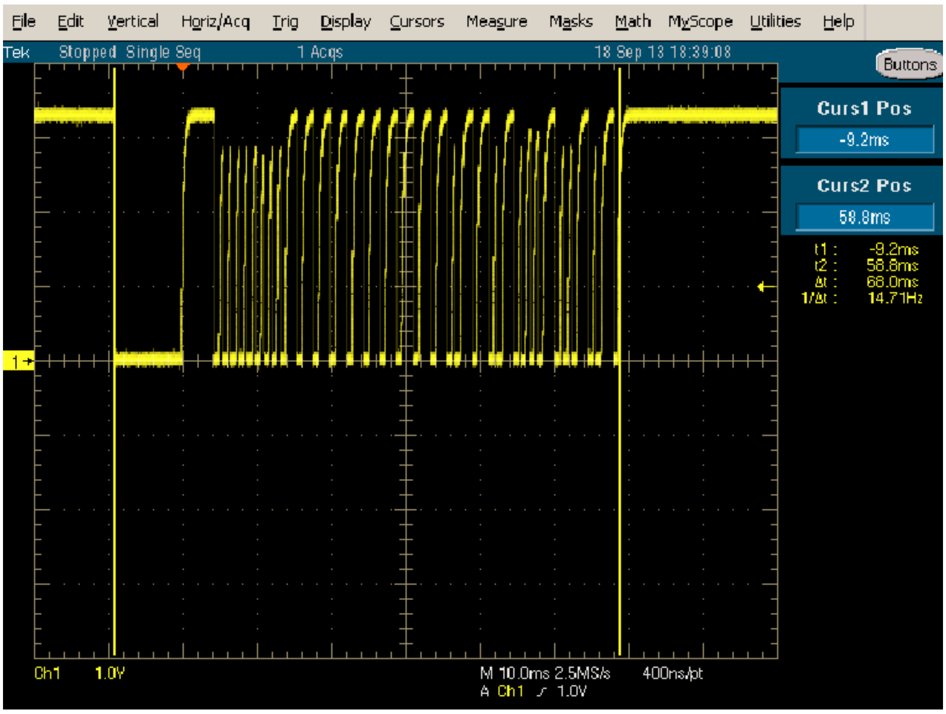
Address = 0000_0000 (LSB to MSB); Address Bar = 1111_1111 (LSB to MSB)

Command = 1101_1010 (LSB to MSB); Command Bar = 0010_0101 (LSB to MSB)


File Edit Vertical Horiz/Acq Trig Display Cursors Measure Masks Math MyScope Utilities Help
Tek Stopped Single Seq 1 Acqs 26 Sep 13 15:17:32
Buttons
1
Ch1 1.0V
M 10.0ms 1.25MS/s 800ns/pt
A Ch1 ⁄ 1.22V

```
Curs1 Pos
0.0s
Curs2 Pos
67.8ms
      t1 :    0.0s
      t2 :    67.8ms
      Δt :    67.8ms
    1/Δt :    14.75Hz
Freq(C2)!      138.9kHz
μ:  138.88889k
m: 138.9k      M: 138.9k
σ: 0.0         n:  1.0
Period(C1)     26.39μs
μ:  26.394513μ
m: 26.39μ      M: 26.39μ
σ: 0.0         n:  1.0
Freq(C1)       37.89kHz
μ:  37.886662k
m: 37.89k      M: 37.89k
σ: 0.0         n:  1.0
```

Figure 7.2 Measure FT234XD TXD pin to GND


File Edit Vertical Horiz/Acq Trig Display Cursors Measure Masks Math MyScope Utilities Help
Buttons
1

Tek Stopped Single Seq 1 Acqs 18 Sep 13 18:39:08

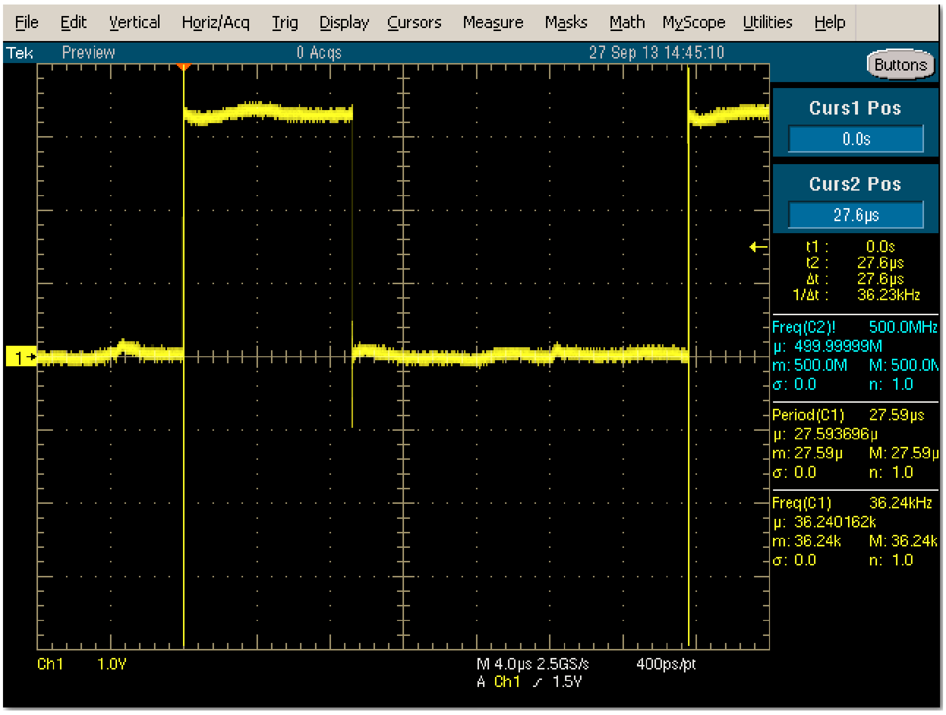
Curs1 Pos -9.2ms

Curs2 Pos 58.8ms

t1 : -9.2ms
t2 : 58.8ms
Δt : 68.0ms
1/Δt : 14.71Hz

Ch1 1.0V M 10.0ms 2.5MS/s 400ns/pt
A Ch1 ⟋ 1.0V

Figure 7.3 Measure the IR receiver


File Edit Vertical Horiz/Acq Trig Display Cursors Measure Masks Math MyScope Utilities Help
Buttons
1

Tek Preview 0 Acqs 27 Sep 13 14:45:10

Curs1 Pos 0.0s

Curs2 Pos 27.6μs

t1 : 0.0s
t2 : 27.6μs
Δt : 27.6μs
1/Δt : 36.23kHz

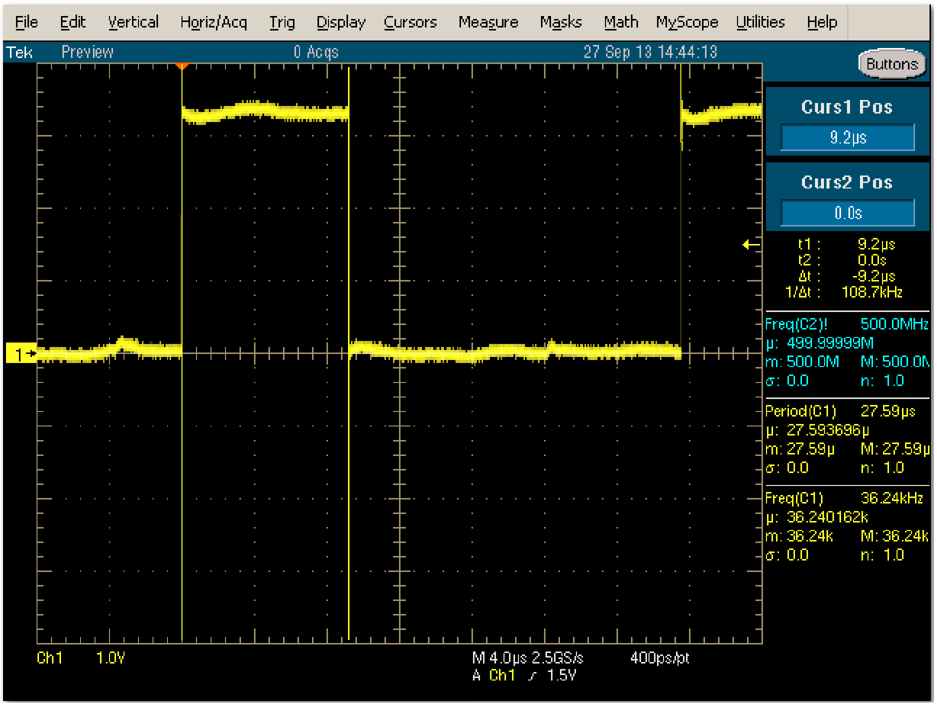
Freq(C2)! 500.0MHz
μ: 499.99999M
m: 500.0M M: 500.0M
σ: 0.0 n: 1.0

Period(C1) 27.59μs
μ: 27.593696μ
m: 27.59μ M: 27.59μ
σ: 0.0 n: 1.0

Freq(C1) 36.24kHz
μ: 36.240162k
m: 36.24k M: 36.24k
σ: 0.0 n: 1.0

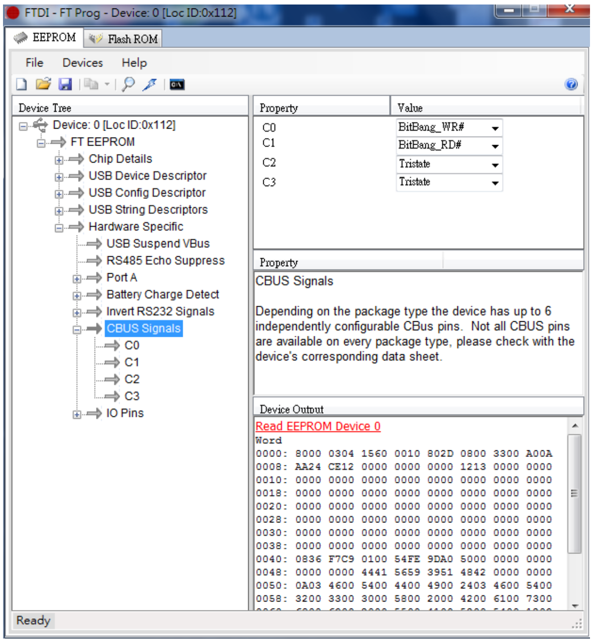
Ch1 1.0V M 4.0μs 2.5GS/s 400ps/pt
A Ch1 ⟋ 1.5V

Figure 7.4 27.6usec for Philips RC-5/ RC-6 carrier (Generated by MyAsyncBitBang.cpp)


File Edit Vertical Horiz/Acq Trig Display Cursors Measure Masks Math MyScope Utilities Help
Tek Preview 0 Acqs 27 Sep 13 14:44:13
Buttons
1
Ch1 1.0V
M 4.0µs 2.5GS/s 400ps/pt
A Ch1 ⟋ 1.5V

```
Curs1 Pos
9.2µs
Curs2 Pos
0.0s
t1 :    9.2µs
t2 :    0.0s
Δt :   -9.2µs
1/Δt : 108.7kHz
Freq(C2)!    500.0MHz
µ: 499.99999M
m: 500.0M    M: 500.0M
σ: 0.0       n: 1.0
Period(C1)   27.59µs
µ: 27.593696µ
m: 27.59µ    M: 27.59µ
σ: 0.0       n: 1.0
Freq(C1)     36.24kHz
µ: 36.240162k
m: 36.24k    M: 36.24k
σ: 0.0       n: 1.0
```

Figure 7.5 9.2usec duty cycle for Philips RC-5/ RC-6 carrier (Generated by MyAsyncBitBang.cpp)



Property	Value
C0	BitBang_WR#
C1	BitBang_RD#
C2	Tristate
C3	Tristate

Property

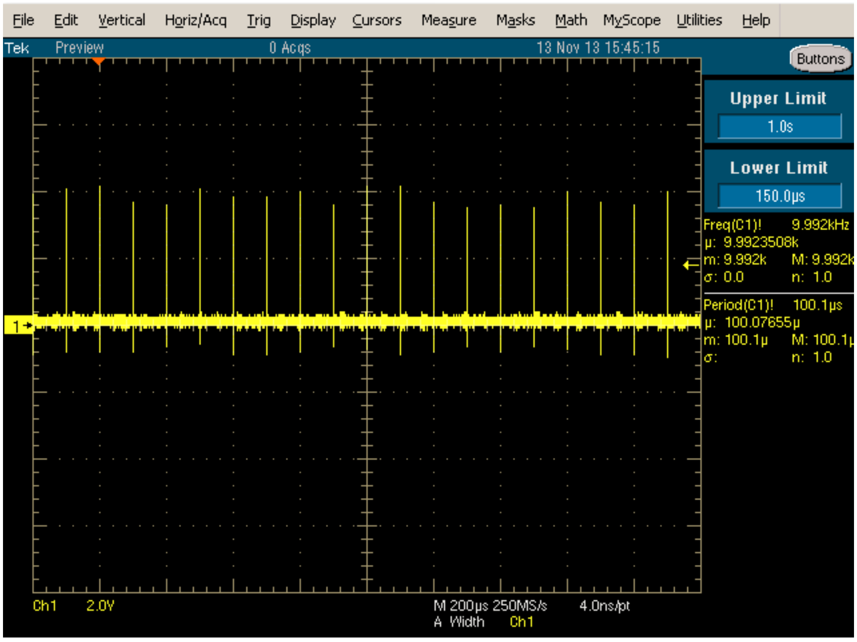
CBUS Signals

Depending on the package type the device has up to 6 independently configurable CBus pins. Not all CBUS pins are available on every package type, please check with the device's corresponding data sheet.

Device Output

```
Read EEPROM Device 0
Word
0000: 8000 0304 1560 0010 802D 0800 3300 A00A
0008: AA24 CE12 0000 0000 0000 1213 0000 0000
0010: 0000 0000 0000 0000 0000 0000 0000 0000
0018: 0000 0000 0000 0000 0000 0000 0000 0000
0020: 0000 0000 0000 0000 0000 0000 0000 0000
0028: 0000 0000 0000 0000 0000 0000 0000 0000
0030: 0000 0000 0000 0000 0000 0000 0000 0000
0038: 0000 0000 0000 0000 0000 0000 0000 0000
0040: 0836 F7C9 0100 54FE 9DA0 5000 0000 0000
0048: 0000 0000 4441 5659 3951 4842 0000 0000
0050: 0A03 4600 5400 4400 4900 2403 4600 5400
0058: 3200 3300 3000 5800 2000 4200 6100 7300
```

Figure 7.6 Set BitBang_WR# on C0 and BitBang_RD# on C1 for a FT-X chip



```
Tek   Preview            0 Acqs            13 Nov 13 15:45:15

Upper Limit
1.0s
Lower Limit
150.0µs

Freq(C1)!      9.992kHz
µ: 9.9923508k
m: 9.992k      M: 9.992k
σ: 0.0         n: 1.0

Period(C1)!    100.1µs
µ: 100.07655µ
m: 100.1µ      M: 100.1µ
σ:             n: 1.0

Ch1   2.0V                 M 200µs 250MS/s   4.0ns/pt
                           A Width   Ch1
```

Figure 7.7 Generate 100 us wavefrom on C1 BitBang_RD#

8 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales)	sales1@ftdichip.com
E-mail (Support)	support1@ftdichip.com
E-mail (General Enquiries)	admin1@ftdichip.com

Branch Office – Tigard, Oregon, USA

Future Technology Devices International Limited
(USA)
7130 SW Fir Loop
Tigard, OR 97223-8160
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales)	us.sales@ftdichip.com
E-Mail (Support)	us.support@ftdichip.com
E-Mail (General Enquiries)	us.admin@ftdichip.com

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited
(Taiwan)
2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan , R.O.C.
Tel: +886 (0) 2 8797 1330
Fax: +886 (0) 2 8751 9737

E-mail (Sales)	tw.sales1@ftdichip.com
E-mail (Support)	tw.support1@ftdichip.com
E-mail (General Enquiries)	tw.admin1@ftdichip.com

Branch Office – Shanghai, China

Future Technology Devices International Limited
(China)
Room 1103, No. 666 West Huaihai Road,
Shanghai, 200052
China
Tel: +86 21 62351596
Fax: +86 21 62351595

E-mail (Sales)	cn.sales@ftdichip.com
E-mail (Support)	cn.support@ftdichip.com
E-mail (General Enquiries)	cn.admin@ftdichip.com

Web Site

http://ftdichip.com

System and equipment manufacturers and designers are responsible to ensure that their systems, and any Future Technology Devices International Ltd (FTDI) devices incorporated in their systems, meet all applicable safety, regulatory and system-level performance requirements. All application-related information in this document (including application descriptions, suggested FTDI devices and other materials) is provided for reference only. While FTDI has taken care to assure it is accurate, this information is subject to customer confirmation, and FTDI disclaims all liability for system designs and for any applications assistance provided by FTDI. Use of FTDI devices in life support and/or safety applications is entirely at the user's risk, and the user agrees to defend, indemnify and hold harmless FTDI from any and all damages, claims, suits or expense resulting from such use. This document is subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Neither the whole nor any part of the information contained in, or the product described in this document, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. Future Technology Devices International Ltd, Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH, United Kingdom. Scotland Registered Company Number: SC136640

Appendix A – References

8.1 Document References

AN_232R-01 for the FT232R and FT245R Bit Bang Modes

AN_233 Java D2xx for Android API User Manual

8.2 Acronyms and Abbreviations

Terms	Description
IR	Infrared
LED	Light Emitting Diodes
OTG	USB On-The-Go
USB	Universal Serial Bus

Appendix B – List of Tables & Figures

8.3 List of Figures

Appendix C – Revision History

Document Title:	AN_260 IR Remote Control Implementation by Java D2XX
Document Reference No.:	FT_000901
Clearance No.:	FTDI# 370
Product Page:	http://www.ftdichip.com/FTProducts.htm
Document Feedback:	Send Feedback

Revision	Changes	Date
1.0	Initial Release	2014-02-17