Sanyo CCB (Computer Control Bus)

by banoub banoub

.In order to understand what is a microcontroller communication protocols is !!!!!!

.you need to get your hand in all kind of IC chips, for example LCD display that use some of communication protocols like I2C SPI, or TV tuner PLL that use I2C TEMP sensor and so on....

.Each communication has its own protocols.

. I get my hand on some IC chips that use some old technology (Sanyo CCB bus)

CCB is a chip-to-chip communication protocol developed by Sanyo.

It is similar to Philips� I2C in its purpose, but much simpler, and it look like SPI in wires connection **Configuration.**

.I get this library from gitHub called Sanyo CCB, I did use it to control the Sanyo radio tuner and Sanyo LCD driver IC and Sanyo Single-Chip Volume And Tone Control System IC.

What is Sanyo (CCB)?

This Computer Control Bus (CCB) is a bus format that is designed to ensure that communication in a system configured with a multiple number of ICs is achieved reliably and economically. It is designed for communication between ICs in equipment and not for communication between products requiring long lines.

The CCB is a single master system, which obviates the need for complex arbitration processing. This reduces the hardware

load significantly, enabling the construction of extremely economical systems. Furthermore, many and varied groups of devices including tuners, electronic volumes, digital signal processors for audio application and display drivers are provided to support a broad spectrum of needs. Using software or serial I/O facilities, the CCB can easily interface with many different kinds of controllers so that no special hardware is required.

Configuration of Serial Data Communication

CCB format based communication between the controller and the peripheral ICs can be achieved in two ways: unidirectional communication (one-way communication between the controller and peripheral ICs) or bidirectional communication (two-way communication between the controller and peripheral ICs and two-way communication between the peripheral ICs).

Unidirectional communication is configured with the three lines of the chip enable signal CE, synchronized clock signal CL and input data signal DI, whereas bidirectional communication is configured with the four lines of the chip enable signal CE, synchronized clock signal CL, input data signal DI and output data signal DO. In recent years, ICs with a new means of bidirectional communication, consisting of the three lines of the chip enable signal CE, synchronized clock signal CL and input/output data signal DI/DO, have also being developed.

Serial Data Communication Format

Under the CCB format, when data is to be sent from the controller to the peripheral ICs, it is sent by the following series of operations. The controller transfers the CCB addresses B0 to B3 and A0 to A3 while the chip enable signal CE is low. After all the addresses have been transferred, the CE signal level is switched from low to high, and while this signal is high, the data DI0 to DIn (input data) required by the peripheral ICs is transferred. The CE signal level is then switched from high to low. In addition, the CCB addresses and input data to be sent from the controller are imported into the peripheral ICs by the rising edge of the CL signal.

On the other hand, when data is to be received by the controller from the peripheral ICs, it is received by the following series of operations. The controller transfers the CCB addresses B0 to B3 and A0 to A3 while the chip enable signal CE is low. After all the addresses have been transferred, the CE signal level is switched from low to high, and while this signal is high, the data DO0 to DOm (output data) is read from the peripheral ICs. CE signal level is then switched from high to low. In addition, since the CCB addresses to be sent from the controller are imported into the peripheral ICs by the rising edge of the CL signal, and the output data to be received by the controller is output from the peripheral ICs in synchronization with the falling edge of the CL signal, the controller can import the output data DO0 to DOm at the rising edge of the CL signal.

As a basic rule, the CCB addresses have a bit width of 8 bits, and the data has a bit width that is a multiple of 8 (8 a bit width where "a" is a natural number). However, there are 4-bit CCB address groups among the CCB address groups so, in this case, dummy data must be added to the CCB addresses B0 to B3.

the Data Read Request Signal and Serial Data Communication

Also provided with the CCB format is a function that passes the data read request signal along the output data signal DO line and outputs it to the controller for cases in which data that has been processed by the peripheral ICs is to be output to the controller, or where the peripheral ICs request that the serial data be sent from the controller, for instance. In addition, when serial data communication is not performed between the controller and peripheral ICs (when the chip enable signal CE level is low), the data read request signal is output by setting the output data signal DO to the low level, and when the controller detects this data read request signal, it proceeds to read the data as shown below for the peripheral ICs.

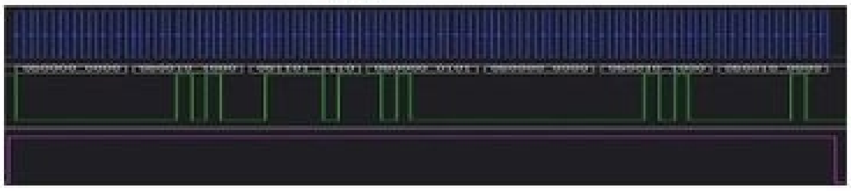
the Interface Pins Connected to the Controller

The chip enable signal input pin CE, synchronized clock signal input pin CL and input data signal input pin DI of the peripheral ICs are input pins that are configured using schmitt circuits, among others. The output data signal output pin DO is an output pin that is configured with an open-drain N-channel transistor, for instance, and a pull-up resistor is required. The electrical characteristics of these pins are as shown below. (For further details on the electrical characteristics of these pins, refer to the individual catalog and delivery specification documents of each device.)

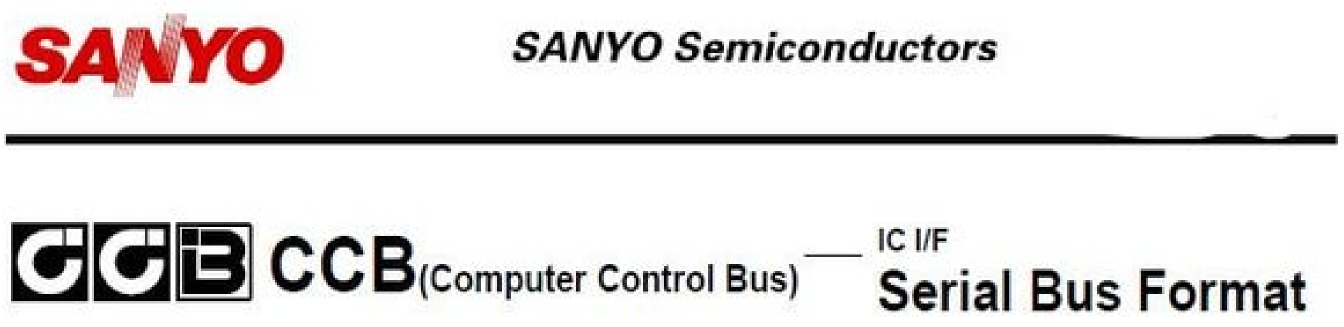
Supplies:

1. I did pull off the LCD display board from Volvo car (1998)
2. Sanyo radio tuner from an old cassette
3. Sanyo Single-Chip Volume and Tone Control System IC from an old cassette
4. I did cut of an audio amplifier board with speaker from an old LCD TV
5. Remote control IR receiver board from an old plasma TV

6. Samsung remote control
7. Arduino Nano
8. old desktop power supply



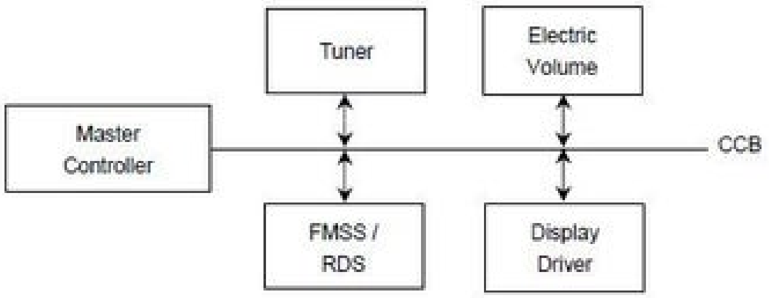
CCB

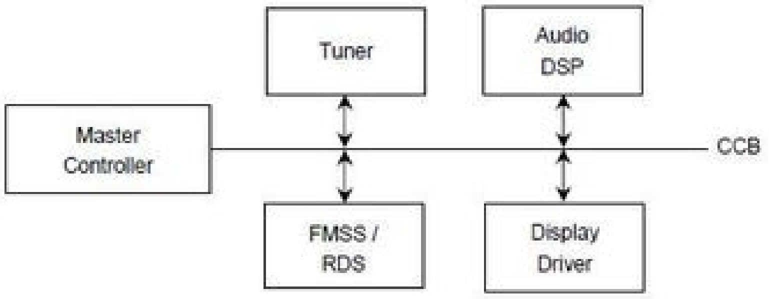


SANYO Semiconductors

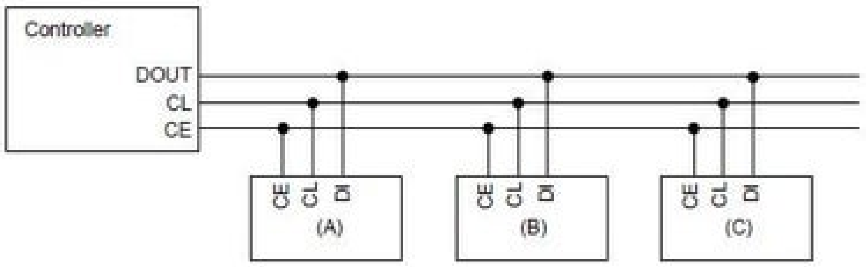
CCB (Computer Control Bus) — IC I/F Serial Bus Format

CCB (Computer Control Bus)



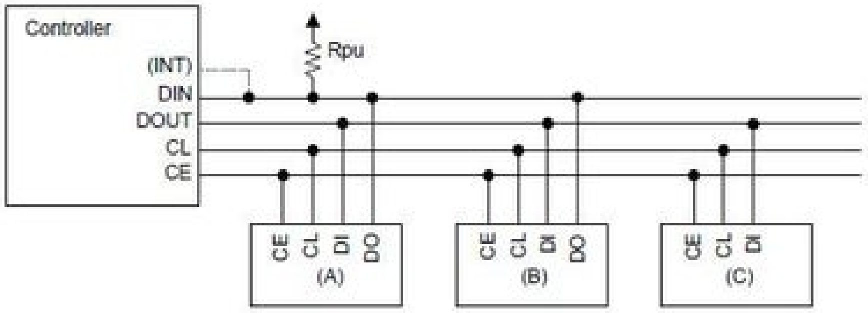


1. Unidirectional communication (one-way communication between the controller and peripheral ICs)
   Configured with the three lines of the chip enable signal CE, synchronized clock signal CL and input data signal DI

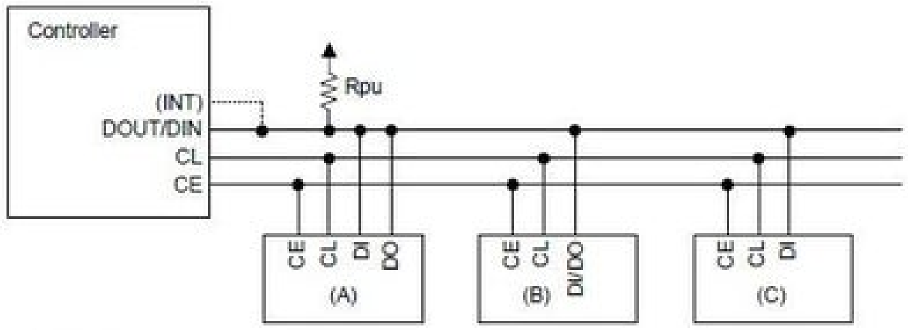


2. Bidirectional communication (two-way communication between the controller and peripheral ICs)

<2-1> Configured with the four lines of the chip enable signal CE, synchronized clock signal CL, input data signal DI and output data signal DO

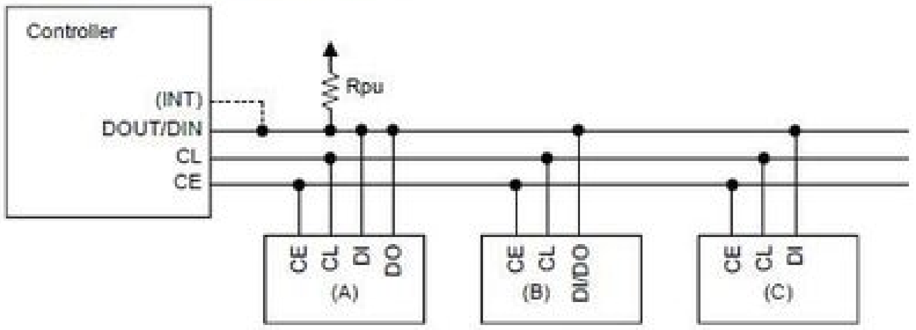


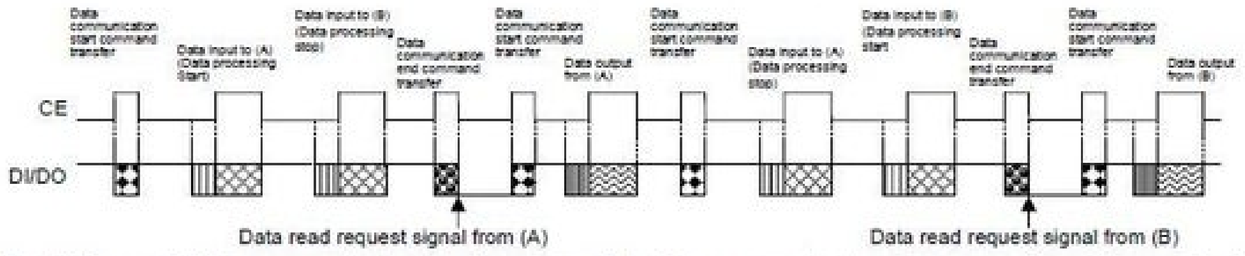
<2-2> Configured with the three lines of the chip enable signal CE, synchronized clock signal CL and input/output data signal DI/DO (This means of bidirectional communication is new for the CCB, and ICs featuring this function must be employed in order to use this communication method.)



[Note] Rpu: Pull-up resistor
(INT): Input port for detecting data read request signal

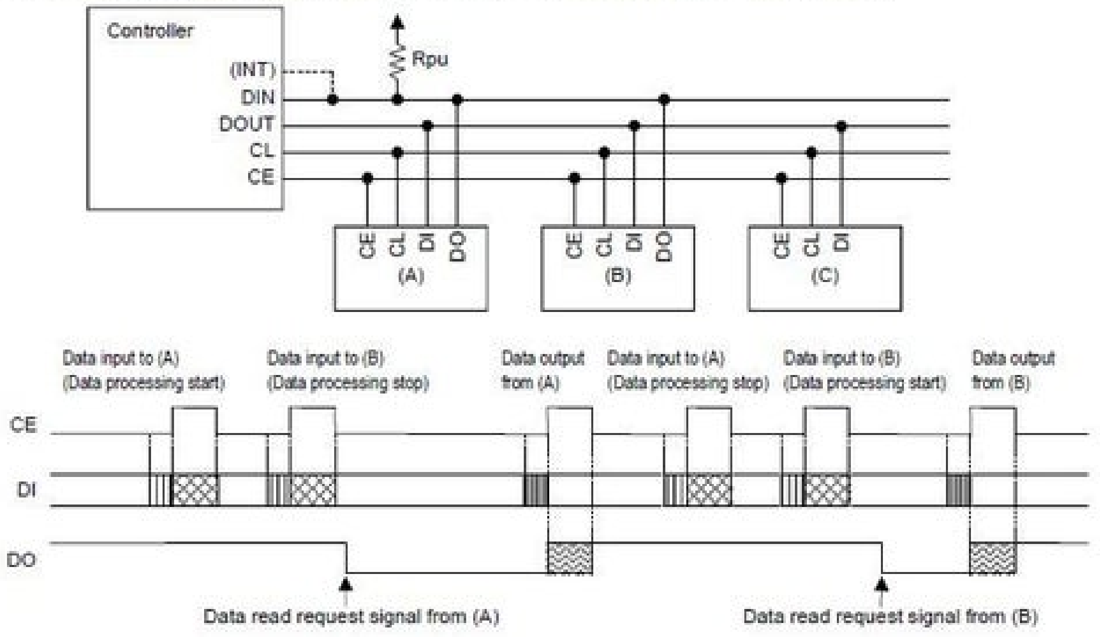
<2> Bidirectional communication format (consisting of the 3 lines of CE, CL, and DI/DO)
(This means of bidirectional communication is new for the CCB, and ICs featuring this function must be employed in order to use this communication method.)



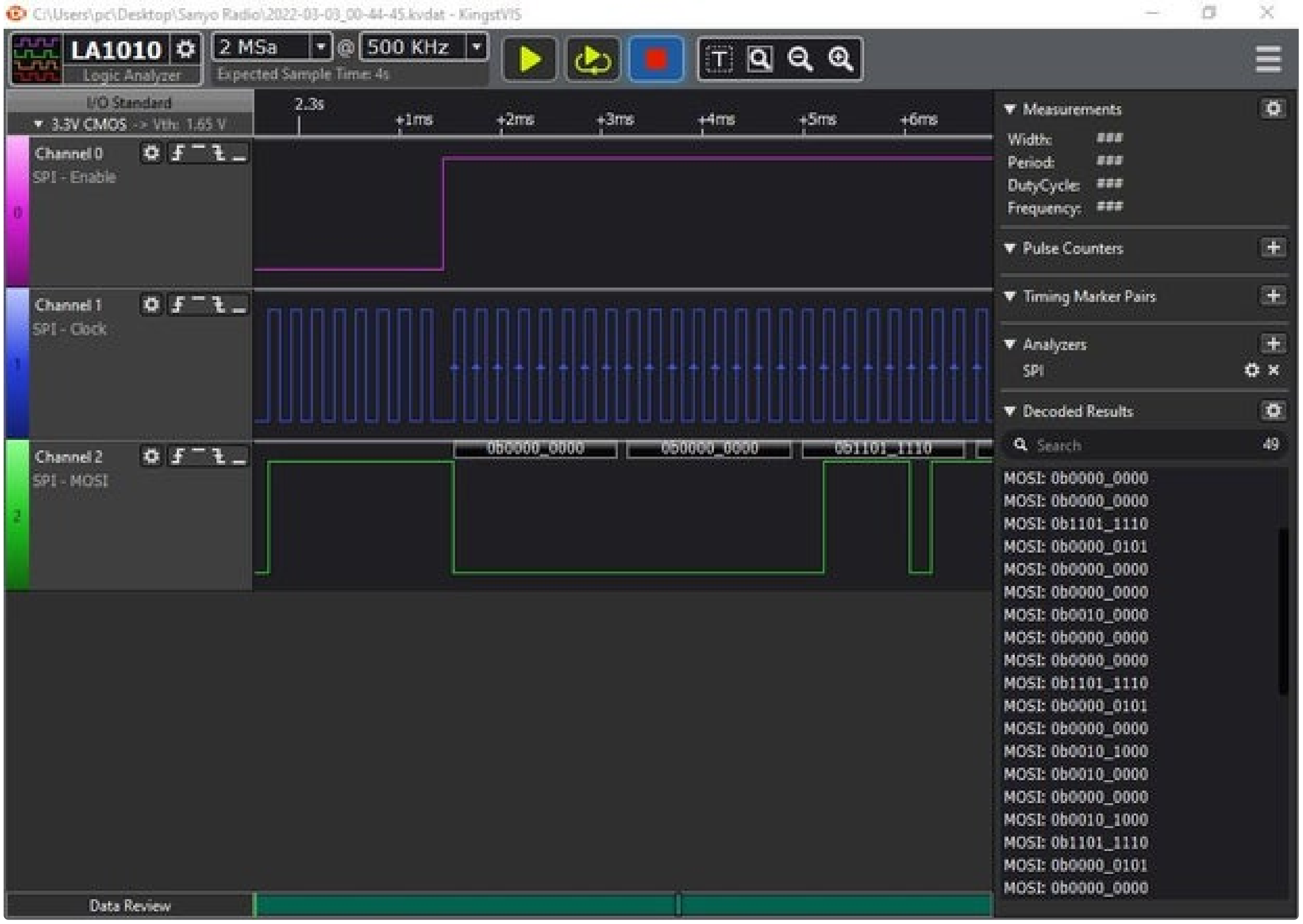

Data communication start command transfer
Data input to (A) (Data processing Start)
Data input to (B) (Data processing stop)
Data communication end command transfer
Data communication start command transfer
Data output from (A)
Data communication start command transfer
Data input to (A) (Data processing stop)
Data input to (B) (Data processing start)
Data communication end command transfer
Data communication start command transfer
Data output from (B)
CE
DI/DO
Data read request signal from (A)
Data read request signal from (B)

[Note] When serial data communication is to be performed, the data communication start command must be transferred. When the data read request signal (DO = low) is to be detected, the data communication end command must be transferred. For further details on the data communication start and end commands, refer to the individual specification document of each device.

<1> Bidirectional communication format (consisting of the 4 lines of CE, CL, DI, and DO)



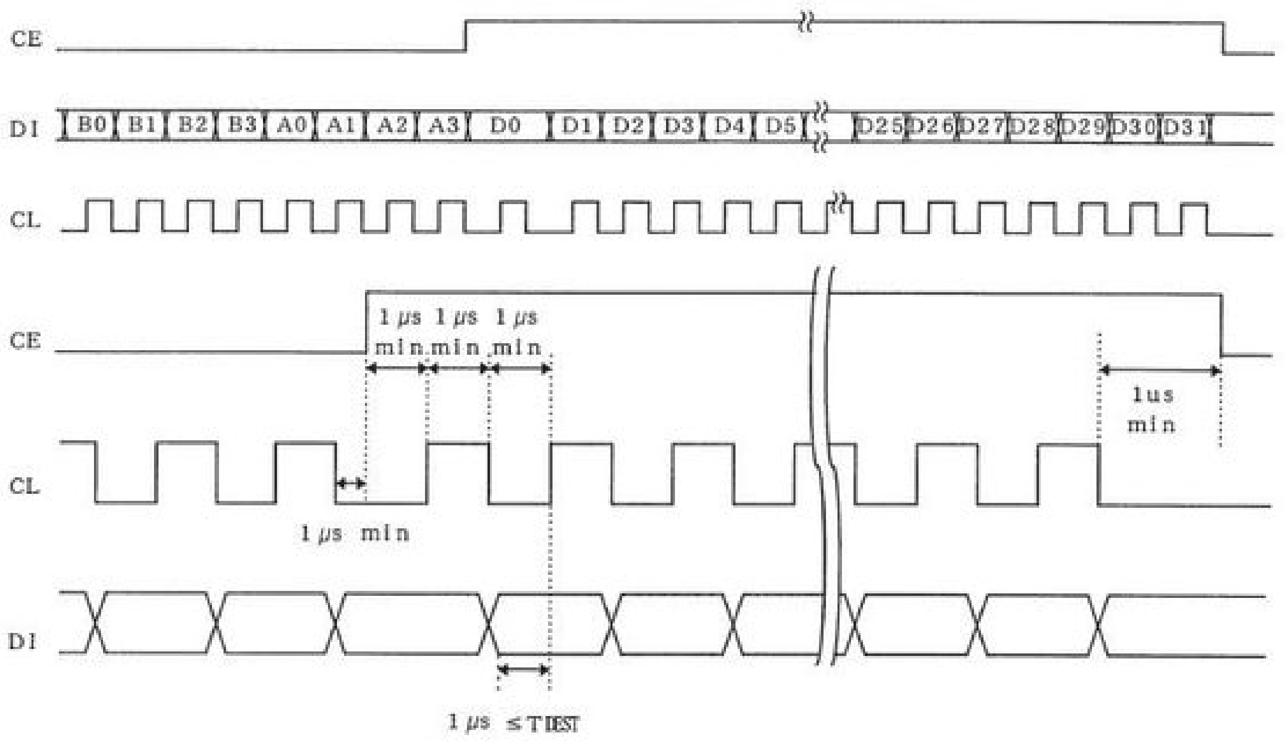
Controller
(INT)
DIN
DOUT
CL
CE
Rpu
CE CL DI DO (A)
CE CL DI DO (B)
CE CL DI (C)
Data input to (A) (Data processing start)
Data input to (B) (Data processing stop)
Data output from (A)
Data input to (A) (Data processing stop)
Data input to (B) (Data processing start)
Data output from (B)
CE
DI
DO
Data read request signal from (A)
Data read request signal from (B)

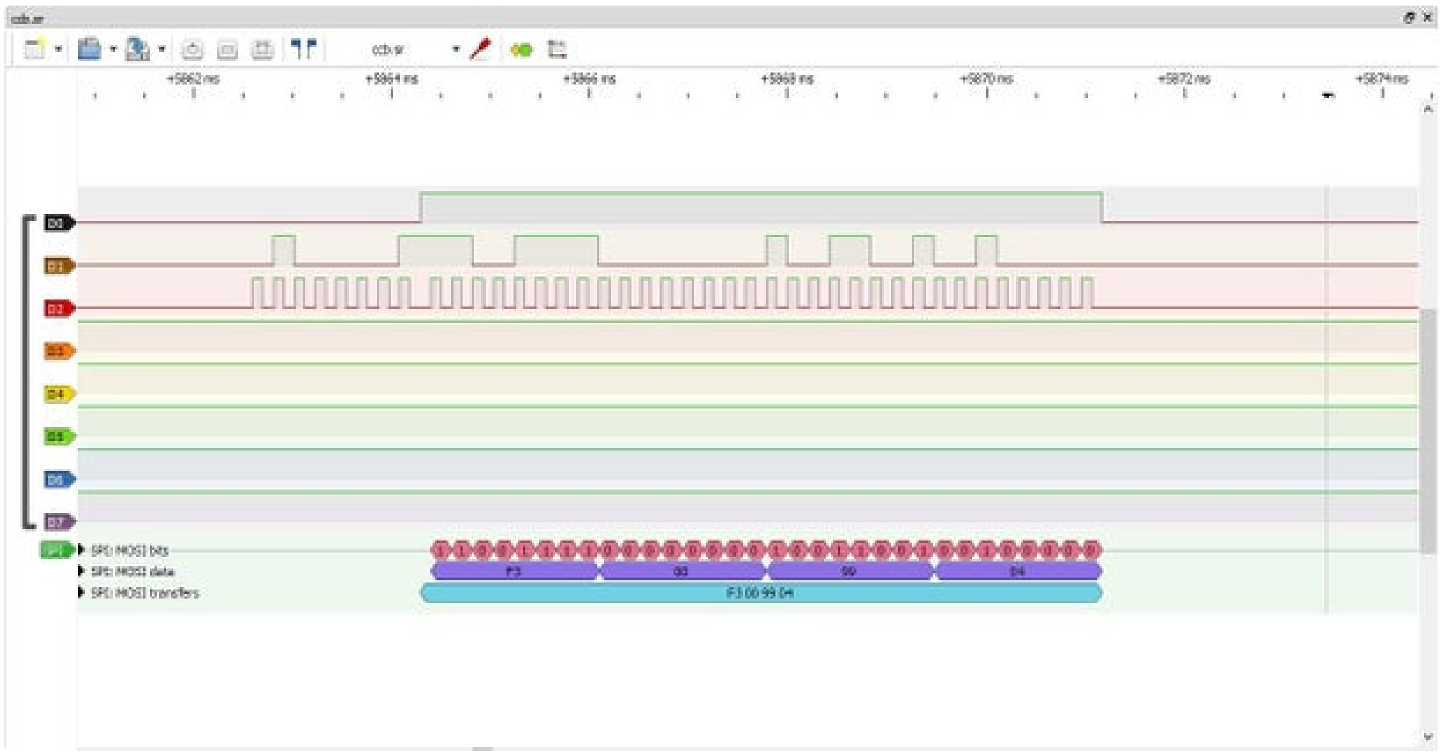


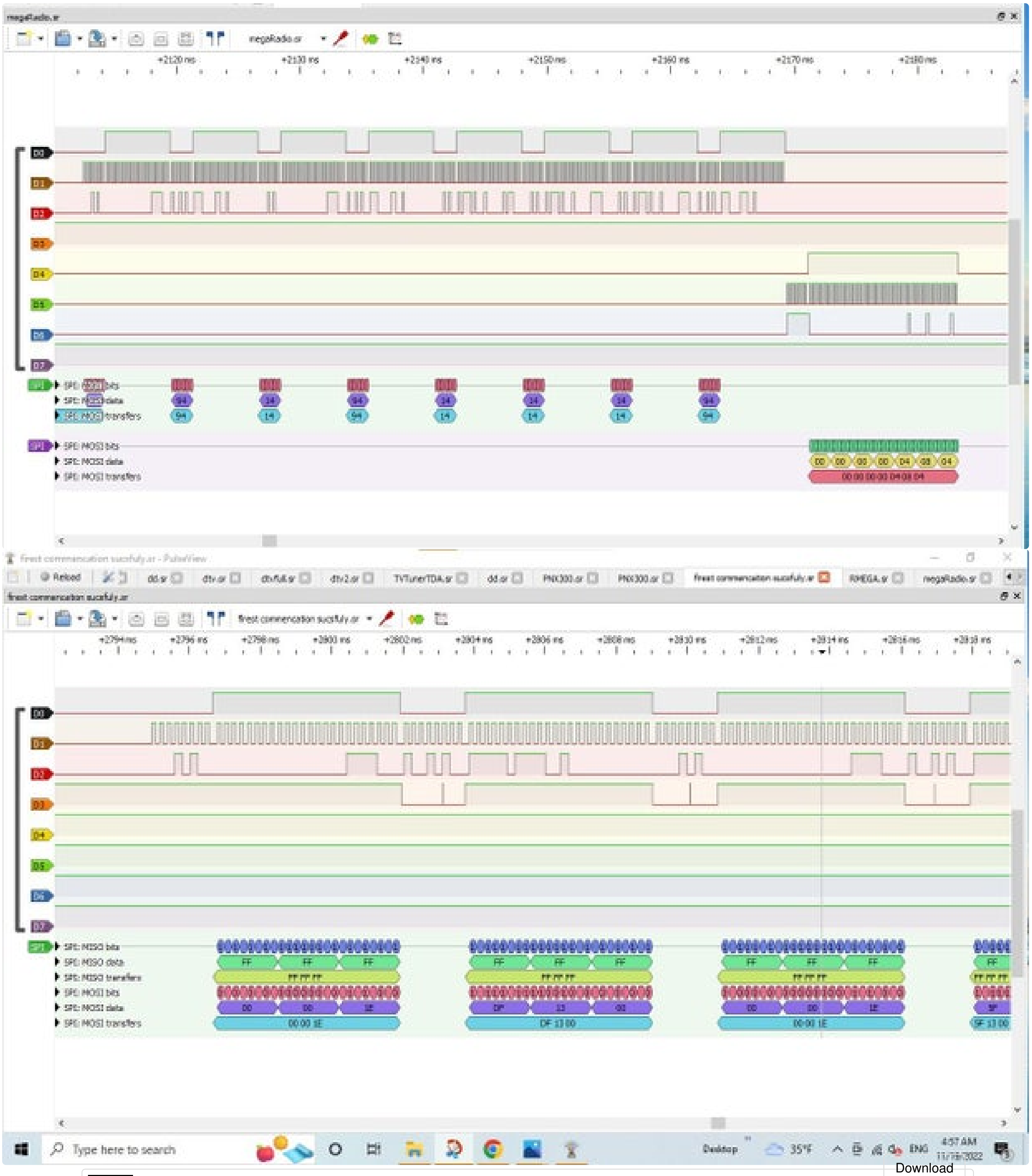
```
MOSI: 0b0000_0000
MOSI: 0b0000_0000
MOSI: 0b1101_1110
MOSI: 0b0000_0101
MOSI: 0b0000_0000
MOSI: 0b0000_0000
MOSI: 0b0010_0000
MOSI: 0b0000_0000
MOSI: 0b0000_0000
MOSI: 0b1101_1110
MOSI: 0b0000_0101
MOSI: 0b0000_0000
MOSI: 0b0010_1000
MOSI: 0b0010_0000
MOSI: 0b0000_0000
MOSI: 0b0010_1000
MOSI: 0b1101_1110
MOSI: 0b0000_0101
MOSI: 0b0000_0000
```

Control System Timing and Data Format

Applications control the LC75342 and LC75342M by applying the stipulated serial data to the CL, DI, and CE pins. This data consists of a total of 40 bits, of which 8 bits are the address and 32 bits are the data itself.


CE
DI
B0 B1 B2 B3 A0 A1 A2 A3 D0 D1 D2 D3 D4 D5 D25 D26 D27 D28 D29 D30 D31
CL
CE
1 μs min
1 μs min
1 μs min
1us min
CL
1 μs min
DI
1 μs ≤ TDEST





Download

[redacted] https://www.instructables.com/FRV/RAY0/LAMIIZW0/FRVRAY0LAMIIZW0.pdf

Step 1: Sanyo LC72131 AM/FM PLL Frequency Synthesizer

.It is very good practice to program different Phase-Locked Loop (PLL) like radio or TV tuner new or old technology, It will help you understand how Frequency demodulation work, I did program three different TV tuner and three different Radio tuner, there is a lot of similarity in (PLL) but different technique.

For communication to work, both the sender and the receiver must agree on what communication channel to use. After which, the sender encodes the message and transmits it to the receiver. Then, the receiver receives the message and decodes it. This holds true to FM: the transmitted FM signal is received and must be demodulated to take the information. This is what FM detectors do.

.What is Frequency Demodulation?

FM demodulation is a key process in the reception of a frequency modulated signal. Once the signal has been received, filtered and amplified, it is necessary to recover the original modulation from the carrier. It is this process that is called demodulation or detection.

There are different types of FM demodulators including:

1. Slope Detector
2. Foster-Seeley Discriminator
3. Ratio Detector
4. Pulse-Averaging Discriminators
5. Quadrature Detectors
6. Phase-Locked Loops

.In this project id did use Sanyo LC72131 AM/FM PLL Frequency Synthesizer.

Functions

• High speed programmable dividers

— FMIN: 10 to 160 MHz ....pulse swallower

(built-in divide-by-two prescaler)

— AMIN: 2 to 40 MHz ....pulse swallower

0.5 to 10 MHz ....direct division

• IF counter

— IFIN: 0.4 to 12 MHz ....AM/FM IF counter

• Reference frequencies

— Twelve selectable frequencies

(4.5 or 7.2 MHz crystal)

1, 3, 5, 9, 10, 3.125, 6.25, 12.5, 15, 25, 50 and 100 kHz

• Phase comparator

— Dead zone control

— Unlock detection circuit

— Deadlock clear circuit

• Built-in MOS transistor for forming an active low-pass

filter

• I/O ports

— Dedicated output ports: 4

— Input or output ports: 2

— Support clock time base output

• Serial data I/O

— Support CCB format communication with the system controller.

• Operating ranges

— Supply voltage.........................4.5 to 5.5 V

— Operating temperature............–40 to +85°C

Programmable Divider Calculation Examples

• FM, 50 kHz steps (DVS = 1, SNS = *, FMIN selected)

FM RF = 90.0 MHz (IF = +10.7 MHz)

FM VCO = 100.7 MHz

PLL fref = 25 kHz (R0 to R1 = 1, R2 to R3 = 0)

100.7 MHz (FM VCO) ¸ 25 kHz (fref) ¸ 2 (FMIN: divide-by-two prescaler) = 2014 > 07DE (HEX)

```
#include<Arduino.h>
#include<SanyoCCB.h>
//LC7213IN1, byte 0
#define IN10_P0 0
#define IN10_P1 1
#define IN10_P2 2
#define IN10_P3 3
#define IN10_P4 4
#define IN10_P5 5
#define IN10_P6 6
#define IN10_P7 7
//LC7213IN1, byte 1
#define IN10_P8 0
#define IN10_P9 1
#define IN10_P10 2
#define IN10_P11 3
#define IN10_P12 4
#define IN10_P13 5
#define IN10_P14 6
#define IN10_P15 7
// LC72131 IN1, byte 2
#define IN10_R3     7
#define IN10_R2     6
```

```
#define IN10_R1     5
#define IN10_R0     4
#define IN10_XS     3
#define IN10_CTE    2
#define IN10_DVS    1
#define IN10_SNS    0
// LC72131 IN2, byte 0
#define IN20_TEST2  7
#define IN20_TEST1  6
#define IN20_TEST0  5
#define IN20_IFS    4
#define IN20_DLC    3
#define IN20_TBC    2
#define IN20_GT1    1
#define IN20_GT0    0

// LC72131 IN2, byte 1
#define IN21_DZ1    7
#define IN21_DZ0    6
#define IN21_UL1    5
#define IN21_UL0    4
#define IN21_DOC2   3
#define IN21_DOC1   2
#define IN21_DOC0   1
#define IN21_DNC    0

// LC72131 IN2, byte 2
#define IN22_BO4    7
#define IN22_BO3    6
#define IN22_BO2    5
#define IN22_BO1    4
#define IN22_IO2    3
#define IN22_IO1    2
#define IN22_IOC2   1
#define IN22_IOC1   0

// LC72131 DO0, byte 0
```

```
#define DO0_IN2    7
#define DO0_IN1    6
#define DO0_UL     4

// For function LC72131SetMode
#define LC72131_MONO    1
#define LC72131_STEREO  2
#define LC72131_MUTE    3
#define LC72131_UNMUTE  4
#define LC72131_BAND_FM 5
#define LC72131_BAND_AM 6

SanyoCCB Radioccb(A2, A3, A1, A0); // Pins: DO CL DI CE

byte pll_in1[3];
byte pll_in2[3];

// Initial frequencies
uint16_t FMFrequency = 949;   // MHz * 10
uint16_t AMFrequency = 53;    // KHZ / 10

uint8_t band = LC72131_BAND_FM;
uint8_t tuned = 0;
//***********************************************
 //            LC72131Init()
 // Initialize the PLL settings vectors with
 //parameters common to booth AM and FM modes
//***********************************************
void LC72131Init() {
  memset(pll_in1, 0, 3);
  memset(pll_in2, 0, 3);
  bitSet(pll_in2[0], IN20_IFS);   // IF counter in normal mode
  bitSet(pll_in2[1], IN21_UL0);   // Phase error detection width = 0us
  bitSet(pll_in2[2], IN22_BO4);   // Mute off / normal tuner mode
}

/***********************************************\
 *          LC72131SetMode()
 * Some predefined setups for the LC72131 module
\**********************************************/
void LC72131SetMode(uint8_t mode) {
 switch(mode) {
   case LC72131_STEREO:
    bitClear(pll_in2[2], IN22_BO3);
    break;

   case LC72131_MONO:
    bitSet(pll_in2[2], IN22_BO3);
    break;

   case LC72131_MUTE:
   bitClear(pll_in2[2], IN22_BO4);
    break;

   case LC72131_UNMUTE:
    bitSet(pll_in2[2], IN22_BO4);
    break;

   case LC72131_BAND_FM:
    band = LC72131_BAND_FM;
   // bitWrite(pll_in1[0], IN10_SNS , 0);
    bitWrite(pll_in1[0], IN10_DVS, 1);
    bitWrite(pll_in1[0], IN10_CTE , 1);
    bitWrite(pll_in1[0], IN10_XS, 1);
    bitWrite(pll_in1[0], IN10_R0,  1);
    bitWrite(pll_in1[0], IN10_R1,  0);
    bitWrite(pll_in1[0], IN10_R2,  0);
    bitWrite(pll_in1[0], IN10_R3,  0);
     bitWrite(pll_in2[2], IN22_IOC1, 1);
     bitWrite(pll_in2[2], IN22_IOC2, 1);
     bitWrite(pll_in2[2], IN22_IO1, 1);
     bitWrite(pll_in2[2], IN22_IO2, 0);
     bitWrite(pll_in2[2], IN22_BO1, 0);
     bitWrite(pll_in2[2], IN22_BO2, 0);
     bitWrite(pll_in2[2], IN22_BO3, 0);
     bitWrite(pll_in2[2], IN22_BO4, 0);
     //
     bitWrite(pll_in2[1], IN21_DNC , 1);
     bitWrite(pll_in2[1], IN21_DOC0 , 1);
     bitWrite(pll_in2[1], IN21_DOC1 , 0);
     bitWrite(pll_in2[1],  IN21_DOC2, 0);
     bitWrite(pll_in2[1], IN21_UL0 , 1);
    bitWrite(pll_in2[1], IN21_UL1 , 0);
    bitWrite(pll_in2[1], IN21_DZ0  , 1);
    bitWrite(pll_in2[1], IN21_DZ1 , 0);
     //
     bitWrite(pll_in2[0],  IN20_GT0   , 0);
     bitWrite(pll_in2[0],  IN20_GT1   , 1);//////////
     bitWrite(pll_in2[0],  IN20_TBC   , 0);
     bitWrite(pll_in2[0],  IN20_DLC   , 0);
     bitWrite(pll_in2[0],   IN20_IFS   , 0);
     bitWrite(pll_in2[0],  IN20_TEST0  , 0);
     bitWrite(pll_in2[0],  IN20_TEST1  , 0);
    bitWrite(pll_in2[0],  IN20_TEST2  , 0);
    break;
   case LC72131_BAND_AM:
    band = LC72131_BAND_AM;
   bitWrite(pll_in1[0], IN10_SNS , 1); //
    bitWrite(pll_in1[0], IN10_DVS, 0); //
    bitWrite(pll_in1[0], IN10_CTE , 0); //
    bitWrite(pll_in1[0], IN10_XS, 0); //
    bitWrite(pll_in1[0], IN10_R0,  0); //
    bitWrite(pll_in1[0], IN10_R1,  0); //
    bitWrite(pll_in1[0], IN10_R2,  0); //
    bitWrite(pll_in1[0], IN10_R3,  1); //
     bitWrite(pll_in2[2], IN22_IOC1, 1);
     bitWrite(pll_in2[2], IN22_IOC2, 1);
     bitWrite(pll_in2[2], IN22_IO1, 1);
     bitWrite(pll_in2[2], IN22_IO2, 0); //
     bitWrite(pll_in2[2], IN22_BO1, 1); //
     bitWrite(pll_in2[2], IN22_BO2, 0); //
```

```
      bitWrite(pll_in2[2], IN22_BO3, 0); //
     bitWrite(pll_in2[2], IN22_BO4, 0);
      //
      bitWrite(pll_in2[1], IN21_DNC , 1); //
      bitWrite(pll_in2[1], IN21_DOC0 , 1); //
      bitWrite(pll_in2[1], IN21_DOC1 , 0); //
      bitWrite(pll_in2[1],  IN21_DOC2, 0); //
      bitWrite(pll_in2[1], IN21_UL0 , 0);
     bitWrite(pll_in2[1], IN21_UL1 , 0);
     bitWrite(pll_in2[1], IN21_DZ0  , 0);
     bitWrite(pll_in2[1], IN21_DZ1 , 1);
      //
      bitWrite(pll_in2[0],  IN20_GT0   , 1);
      bitWrite(pll_in2[0],  IN20_GT1   , 0);
      bitWrite(pll_in2[0],  IN20_TBC   , 0);
      bitWrite(pll_in2[0],  IN20_DLC   , 0);
      bitWrite(pll_in2[0],   IN20_IFS   , 0);
      bitWrite(pll_in2[0],  IN20_TEST0  , 0);
      bitWrite(pll_in2[0],  IN20_TEST1  , 0);
     bitWrite(pll_in2[0],  IN20_TEST2  , 0);
    break;
  }
  Radioccb.write(B10000010, pll_in1, 3 );
  Radioccb.write(B10010010, pll_in2, 3);
}




/***********************************************************\
                  LC72131Tune()
  Set the tuner frequency and return 1 if it is tuned
  or 0 otherwise.
  The frequency divisors was chosen in a way the frequency
  representation can be directly sent to the PLL and is
  easy to represent:
   FM mode (divisor = 100): frequency (MHz) * 10
   AM mode (divisor = 10):  frequency (kHZ) / 10
```

```
\**********************************************************/
uint8_t LC72131Tune(uint16_t frequency) {
  uint16_t fpd = 0;

 // unsigned long IFCounter = 0;
  switch(band) {
    case LC72131_BAND_FM:
      // FM: fpd = (frequency + FI) / (50 * 2)
      fpd = (frequency + 107);
      break;
    case LC72131_BAND_AM:
      // AM: fpd = ((frequency + FI) / 10) << 4
      fpd = (frequency + 45) << 4;
      break;
    default: return 1;
  }

  LC72131SetMode(LC72131_MUTE);   // LC72131 only injects FI signal into the PLL when in MUTE mode

  // Reset the IF counter and program the Frequency Programmable Divider (fpd)
  bitClear(pll_in1[0], IN10_CTE);
  pll_in1[1] = byte(fpd >> 8);

  pll_in1[2] = byte(fpd & 0x00ff);

Radioccb.write(B10000010, pll_in1, 3 );

  // Start the IF counter
  bitSet(pll_in1[0], IN10_CTE);
Radioccb.write(B10000010, pll_in1, 3 );



LC72131SetMode(LC72131_UNMUTE);   // Mute off / normal tuner mode



  return 0;
}
```

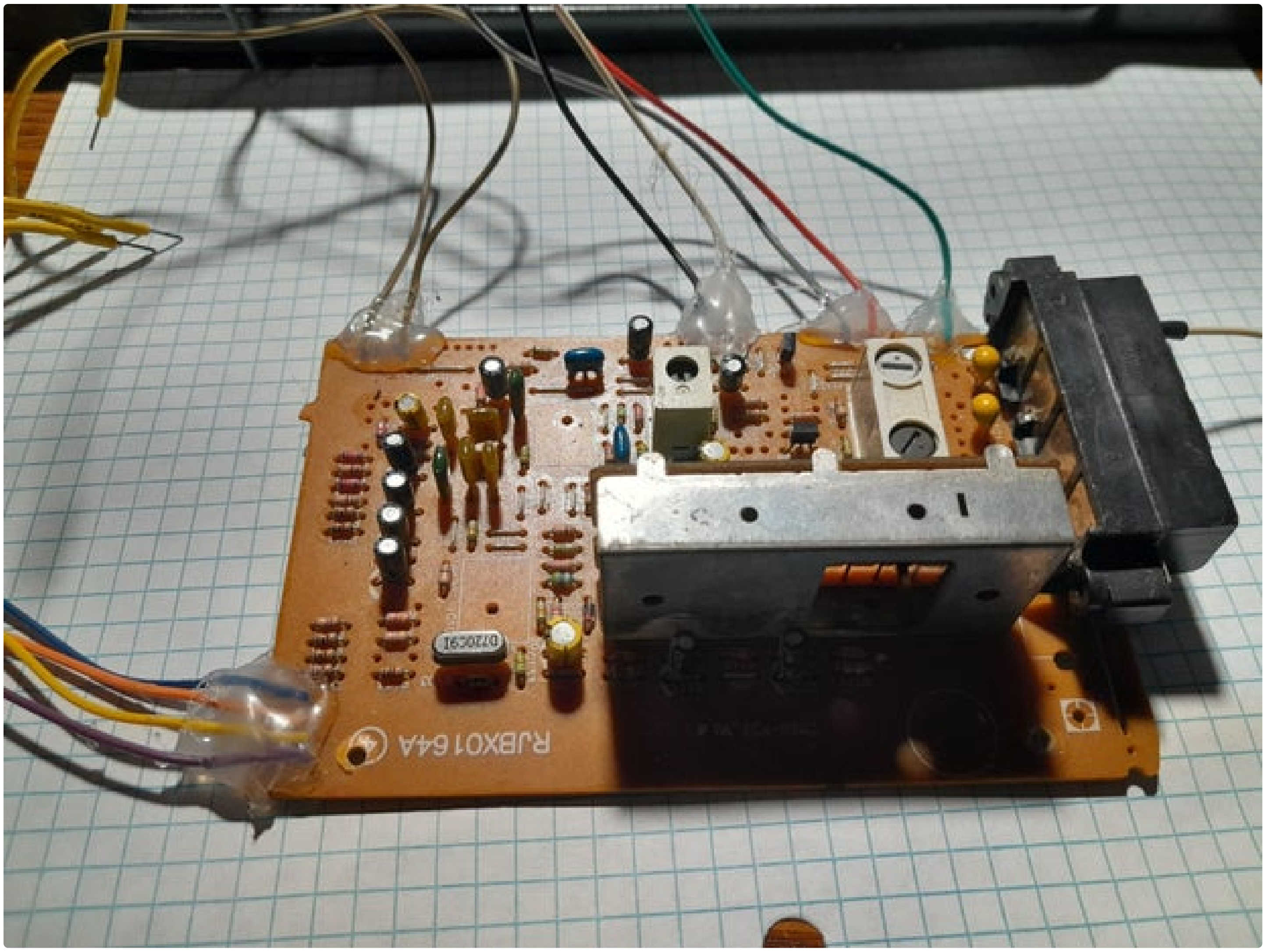
.and I did use Sanyo LA1833N System-on-Chip Tuner IC for Home Stereo Systems.

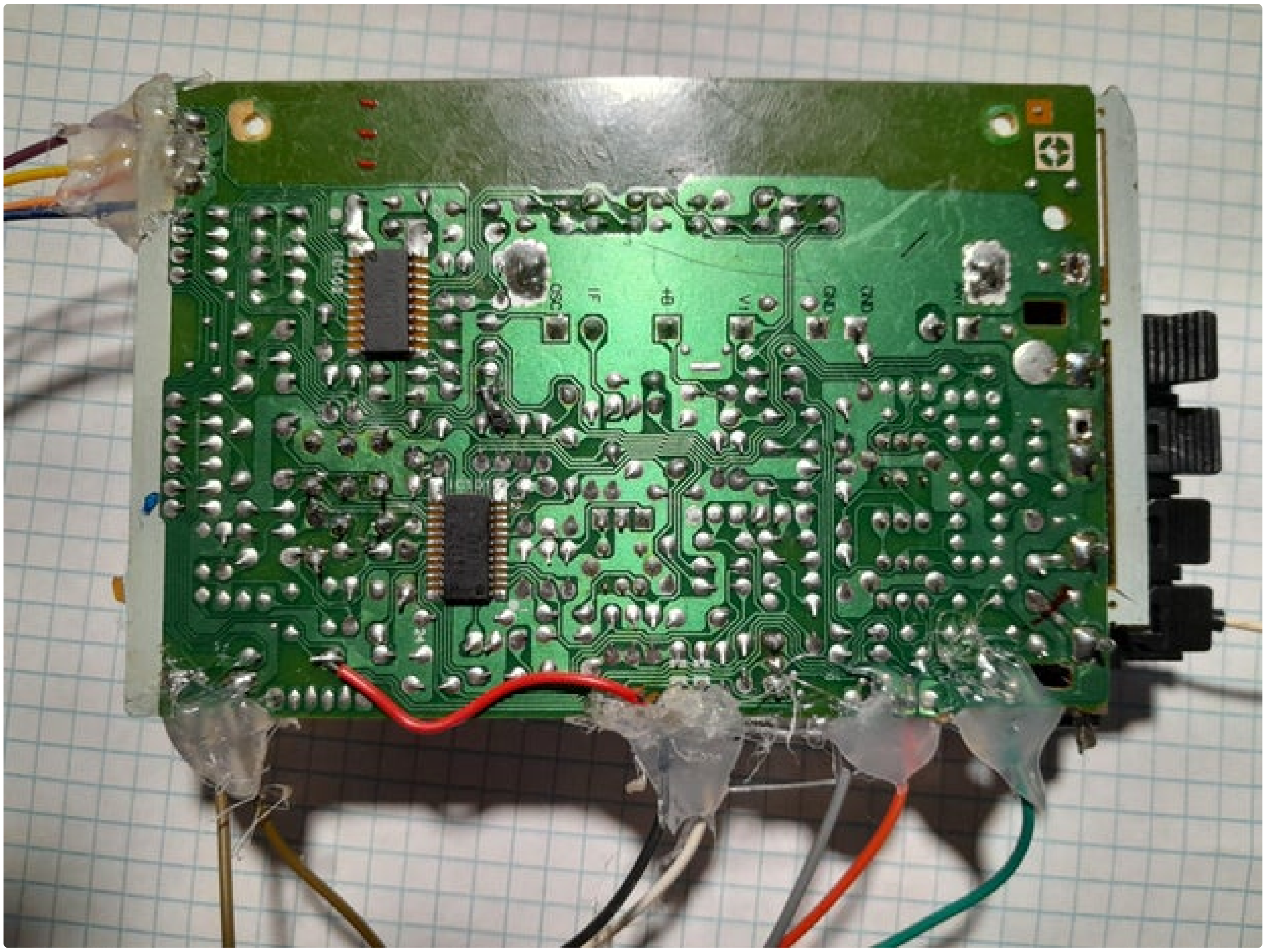
Functions

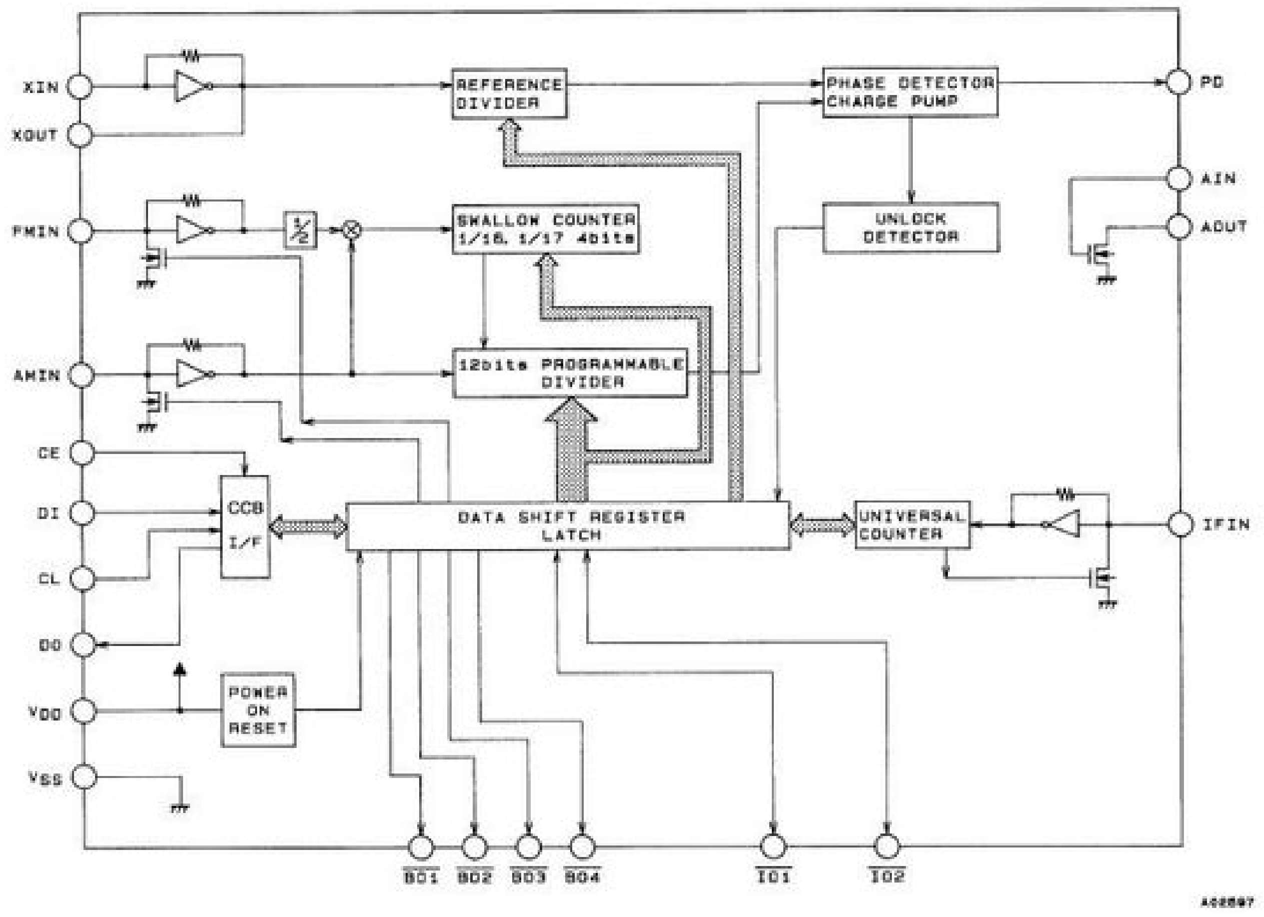
• AM: RF amplifier, mixer, oscillator, IF amplifier,
detector, AGC, SD, oscillator buffer, IF buffer, and
stereo IF output

• FM IF: IF amplifier, **quadrature detector**, S meter, SD,
S-curve detector, IF buffer

• MPX: PLL stereo decoder, stereo indicator, forced
mono, VCO stop, audio muting, adjacent channel
interference reduction function, pilot canceller

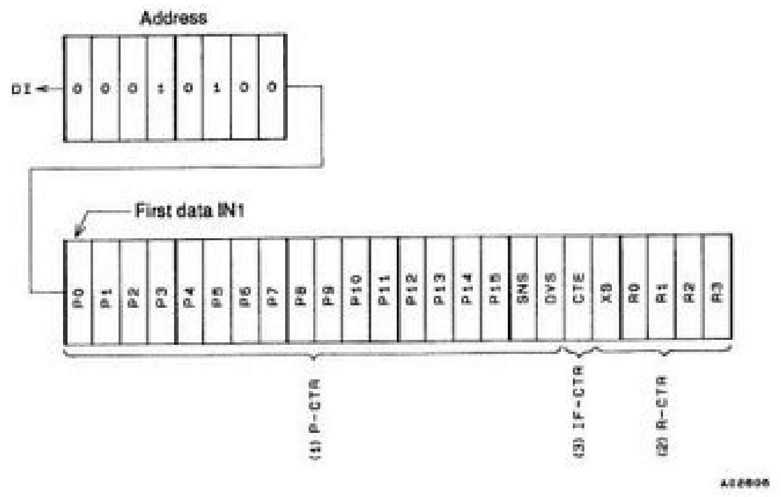

RJBX0164A
D720C9I



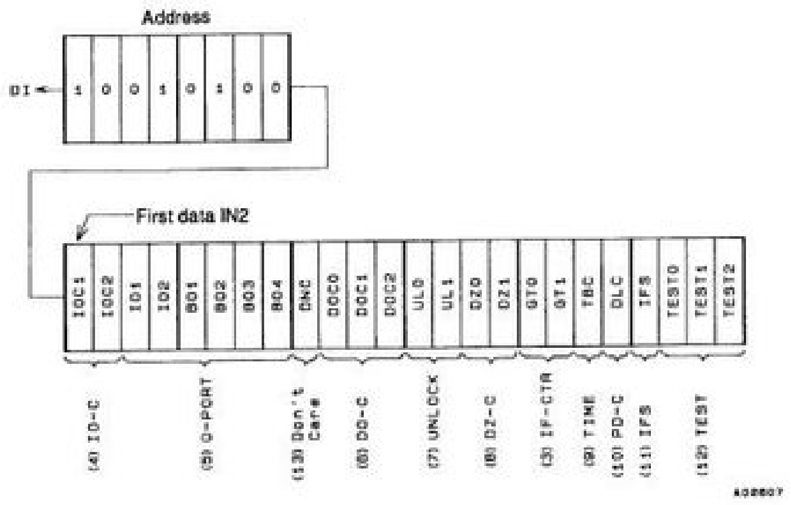


1. DI Control Data (Serial Data Input) Structure

• IN1 Mode


Address
DI
0 0 0 1 0 1 0 0
First data IN1
P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15 SNS DVS CTE XS R0 R1 R2 R3
(1) P-CTR
(3) IF-CTR
(2) R-CTR
A02608

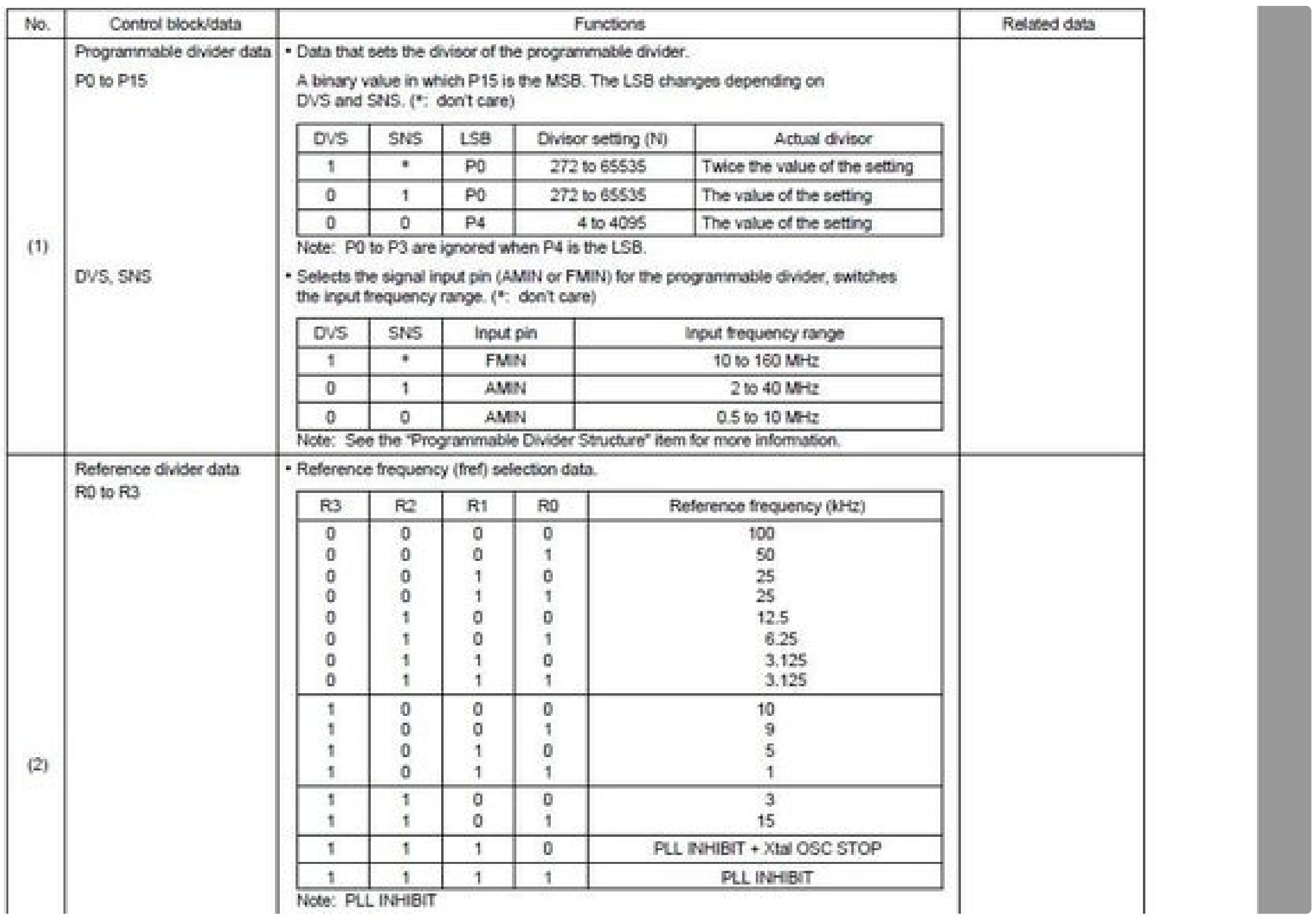
• IN2 Mode



Serial Data I/O Methods

The LC72131 inputs and outputs data using the Sanyo CCB (computer control bus) audio LSI serial bus format. This LSI adopts an 8-bit address format CCB.

	I/O mode	Address								Function
		B0	B1	B2	B3	A0	A1	A2	A3	
1	IN1 (82)	0	0	0	1	0	1	0	0	• Control data input mode (serial data input) • 24 data bits are input. • See the "DI Control Data (serial data input) Structure" item for details on the meaning of the input data.
2	IN2 (92)	1	0	0	1	0	1	0	0	• Control data input mode (serial data input) • 24 data bits are input. • See the "DI Control Data (serial data input) Structure" item for details on the meaning of the input data.
3	OUT (A2)	0	1	0	1	0	1	0	0	Data output mode (serial data output) • The number of bits output is equal to the number of clock cycles. • See the "DO Output Data (serial data output) Structure" item for details on the meaning of the output data.



No.	Control block/data	Functions	Related data
(1)	Programmable divider data P0 to P15	• Data that sets the divisor of the programmable divider. A binary value in which P15 is the MSB. The LSB changes depending on DVS and SNS. (*: don't care)	

DVS	SNS	LSB	Divisor setting (N)	Actual divisor
1	*	P0	272 to 65535	Twice the value of the setting
0	1	P0	272 to 65535	The value of the setting
0	0	P4	4 to 4095	The value of the setting

Note: P0 to P3 are ignored when P4 is the LSB.

DVS, SNS

• Selects the signal input pin (AMIN or FMIN) for the programmable divider, switches the input frequency range. (*: don't care)

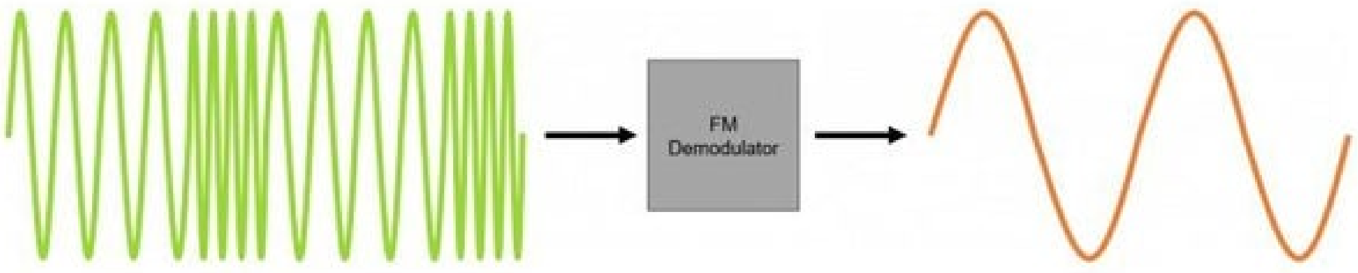
DVS	SNS	Input pin	Input frequency range
1	*	FMIN	10 to 160 MHz
0	1	AMIN	2 to 40 MHz
0	0	AMIN	0.5 to 10 MHz

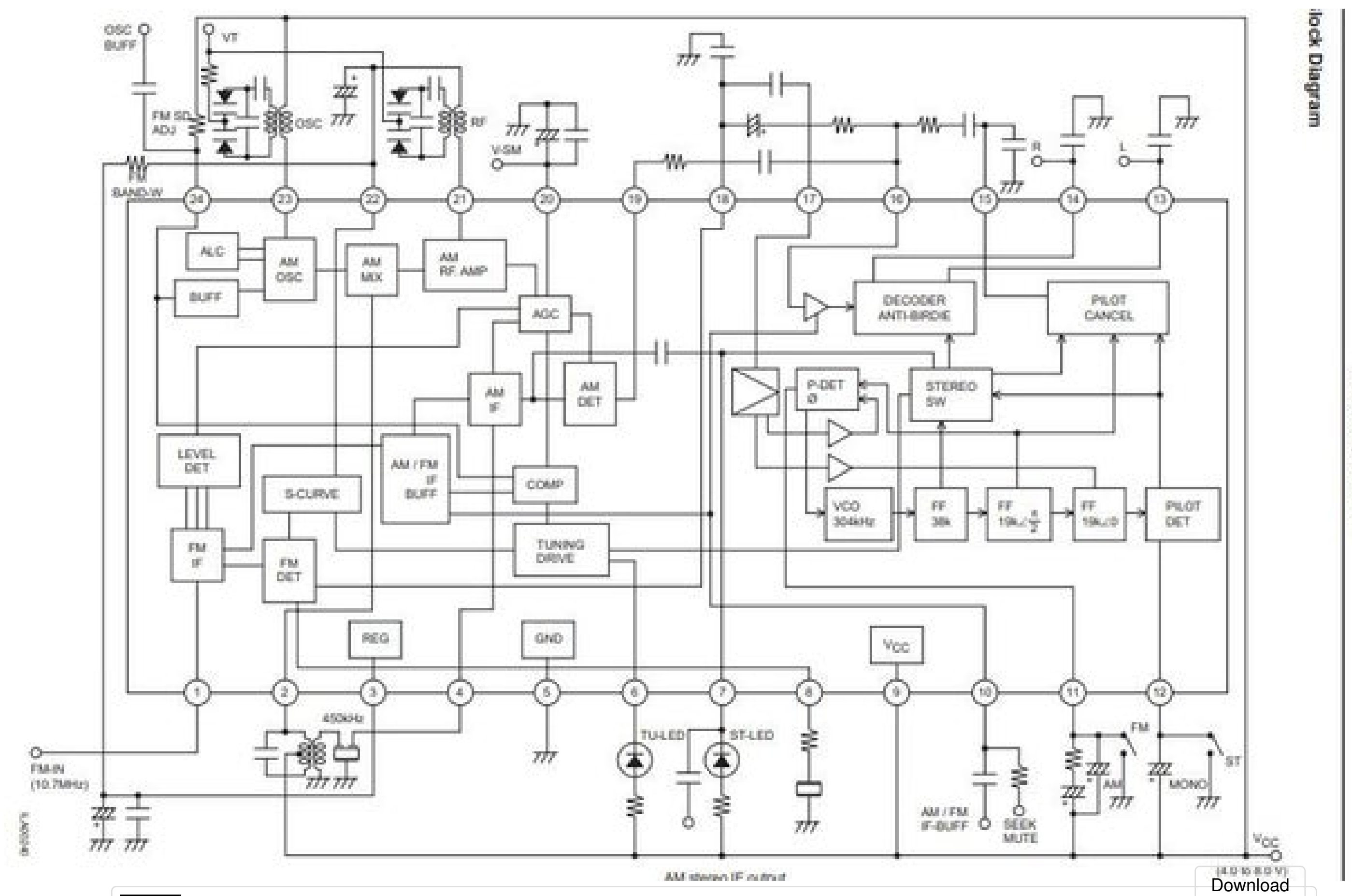
Note: See the "Programmable Divider Structure" item for more information.

No.	Control block/data	Functions	Related data
(2)	Reference divider data R0 to R3	• Reference frequency (fref) selection data.	

R3	R2	R1	R0	Reference frequency (kHz)
0	0	0	0	100
0	0	0	1	50
0	0	1	0	25
0	0	1	1	25
0	1	0	0	12.5
0	1	0	1	6.25
0	1	1	0	3.125
0	1	1	1	3.125
1	0	0	0	10
1	0	0	1	9
1	0	1	0	5
1	0	1	1	1
1	1	0	0	3
1	1	0	1	15
1	1	1	0	PLL INHIBIT + Xtal OSC STOP
1	1	1	1	PLL INHIBIT

Note: PLL INHIBIT





Download

https://www.instructables.com/FPD/22FX/LAMIIZVD/FPD22FXLAMIIZVD.pdf

Download

https://www.instructables.com/FU9/XODB/LAMIIZW2/FU9XODBLAMIIZW2.pdf

Step 2: Sanyo LC75821E LCD Display Drivers

.This display is (TN) LCD

What does TN stand for in LCD?

A **twisted nematic** (TN) display is a common type of liquid-crystal display (LCD) that consists of a substance called a nematic liquid crystal that is confined between two plates of polarized glass

.In my case this display has 3 Digits 7 Segments and small digit 7 segments and one digit 9 segments and 8 static icons and : .

For the Sanyo LC75821E LCD Display Drivers:

The LC75821E and LC75821W are general-purpose LCD

display drivers that can be used for frequency display in

microprocessor-controlled radio receivers and in other

display applications.

Features

• 53 segment outputs (the maximum for static drive)

• Two drive types: static (1/1) duty (53 segments) and 1/2

duty (104 segments)

• Data input: 3 serial input pins

• INH pin for turning off all display output

.First I did practice sending data over Sanyo CCB Bus according to the data sheet of the Sanyo LC75821E LCD Display Drivers, to know which bit control which segments and icons .

.So for this display it is type of Static (1/1)duty

Transfer direction (56 bits)

<----------------------------- D1 D2 D3 ....D47 D48 D49 D50 D51 D52 D53 D0 0 0.

. I have to use bitwrite() Function to select which bit to be 0 or 1, that is a lot of work because the Sanyo LC75821E LCD Display Drivers it need 56 bits to drive the display.

For example to display number one in the first digit from the left of the display it well be

```
#include<Arduino.h>

#include<SanyoCCB.h>

SanyoCCB LCDDisplay(A11, A10, 0, A9);// Pins: DO CL CE

#define   D0 0

#define   D1 1

#define   D2 2

#define   D3 3

#define   D4 4

#define   D5 5

#define   D6 6

#define   D7 7

#define   INH A8

byte Data[7];

void Digit_1(uint8_t number)

{

switch(number)

{

  case 1:

     bitWrite(Data[3],D1,0);

     bitWrite(Data[3],D0,1);

     bitWrite(Data[3],D2,1);

     bitWrite(Data[3],D4,0);

     bitWrite(Data[3],D6,0);

     bitWrite(Data[3],D7,0);

     bitWrite(Data[3],D3,0);

     bitWrite(Data[3],D5,0);
break;
   LCDDisplay.write(B11111111,Data,7);// The first parameter is the chip address, the parameter is the data that need to be send, the third parameter is who many array of that data.
   digitalWrite(INH,HIGH);
}
```

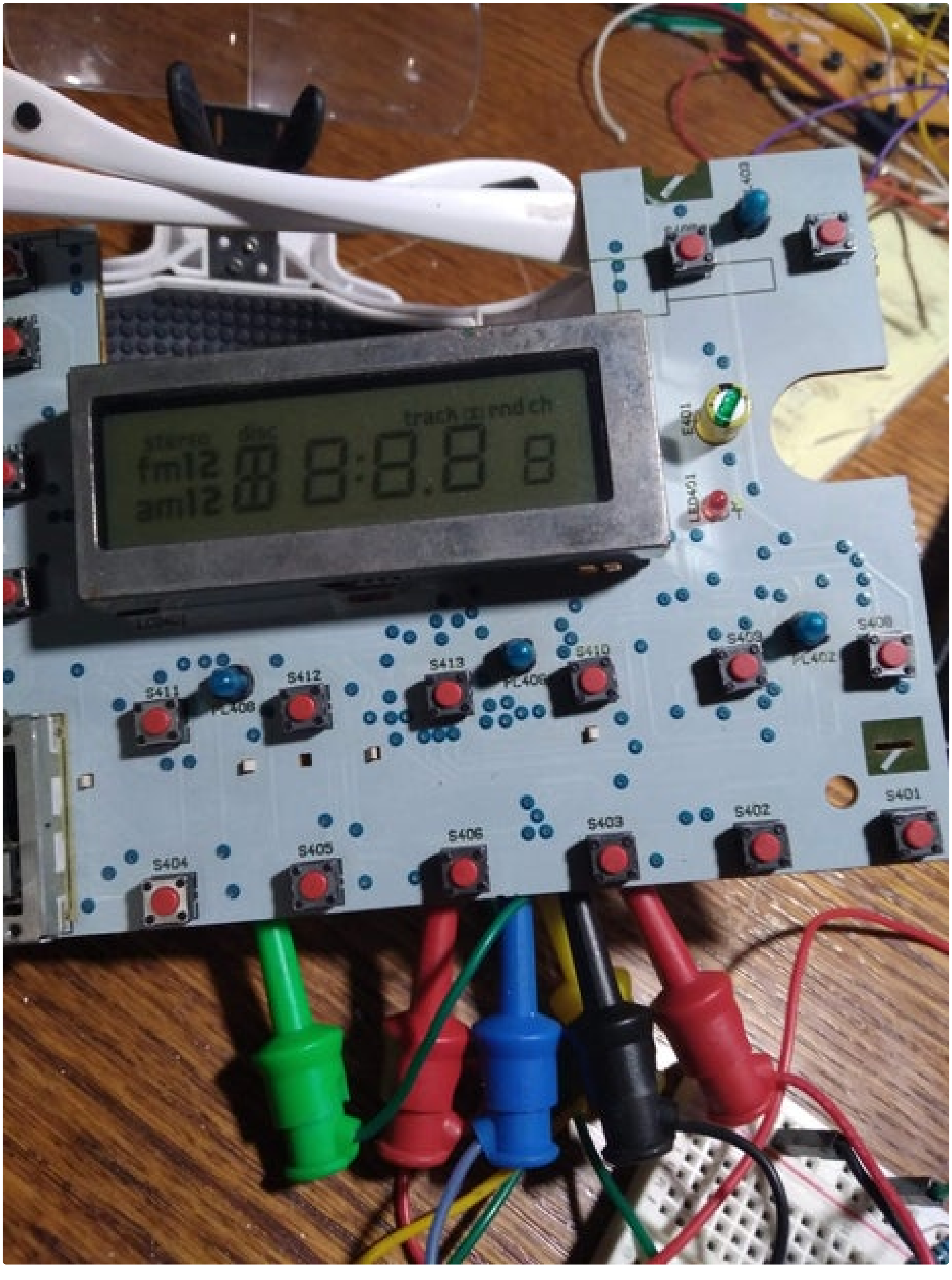
and for number one in the second digit is

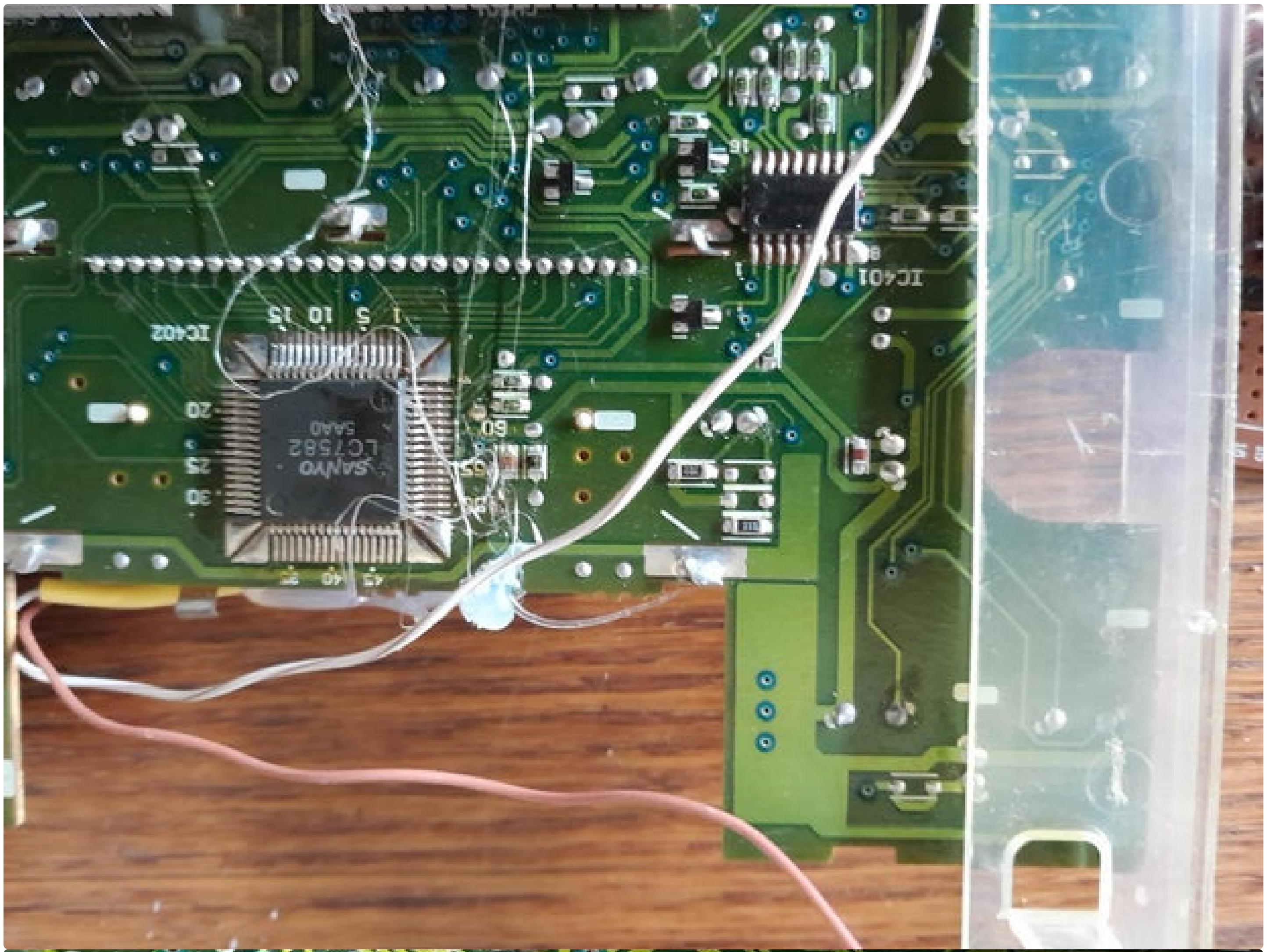
```
void Digit_2(uint8_t number)
{
switch(number)
{
case 1:
bitWrite(Data[4],D7, 0);
bitWrite(Data[4],D1, 1);
bitWrite(Data[4],D3, 1);
bitWrite(Data[4],D2, 0);
bitWrite(Data[4],D4, 0);
bitWrite(Data[4],D6, 0);
bitWrite(Data[4],D5, 0);
break;
LCDDisplay.write(B11111111,Data,7);// The first parameter is the chip address, the parameter is the data that need to be send, the third parameter is who many array of that data.
   digitalWrite(INH,HIGH);
}
```

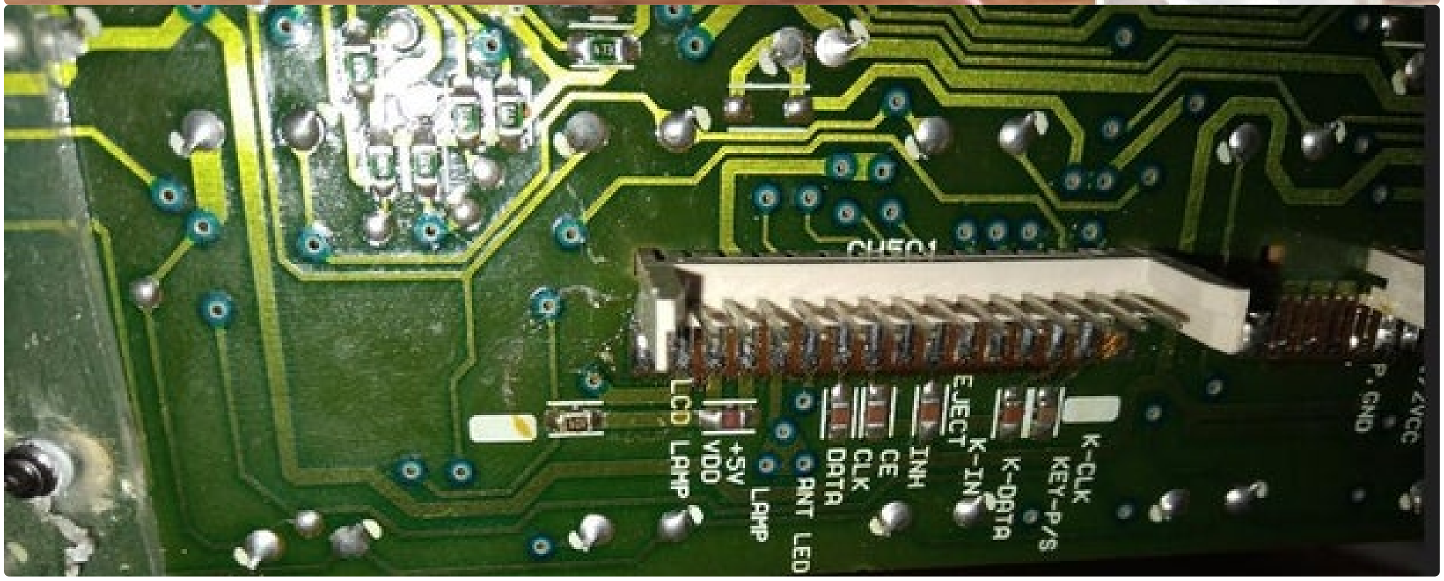
and so on for each digit plus the data for each number from 0 to 9,

(Control code allocation)

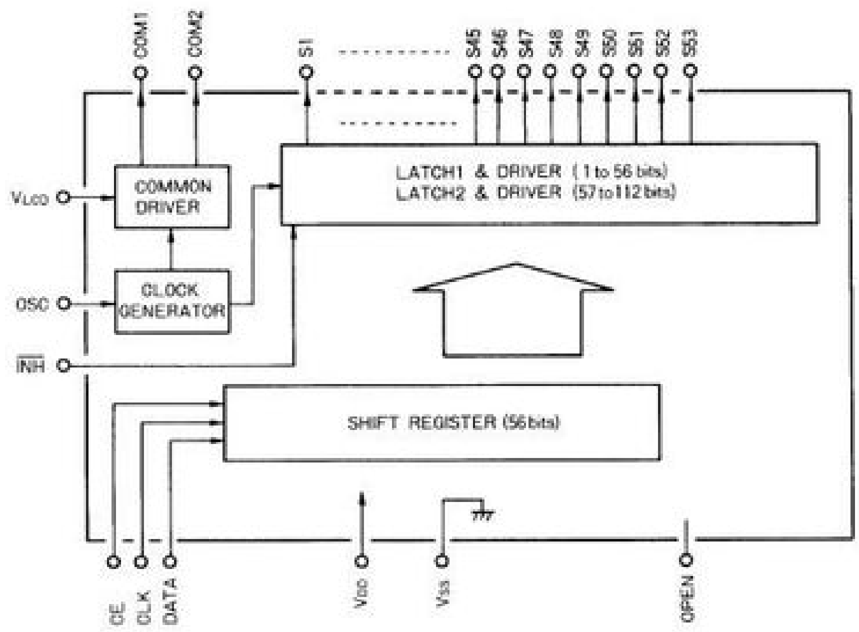
the digit number one from left is from bit number 17 to bit number 25 , and for digit number two it is from bit number 26 to bit number 34. and so on for the rest of the display.


stereo
disc
track
rnd ch
fm12
am12
8:8.8 8
S411
S412
S413
S410
S409
S408
S404
S405
S406
S403
S402
S401
E401
LED401


SANYO
LC7582
5AA0
IC402
IC401

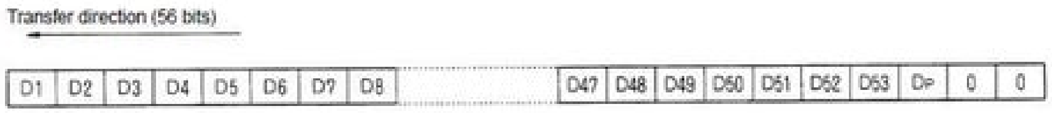

CN501
LCD LAMP
VDD
+5V
LAMP
ANT LED
DATA
CLK
CE
INH
EJECT
K-IN
K-DATA
K-CLK
KEY-P/S
P.GND
VCC

Block Diagram

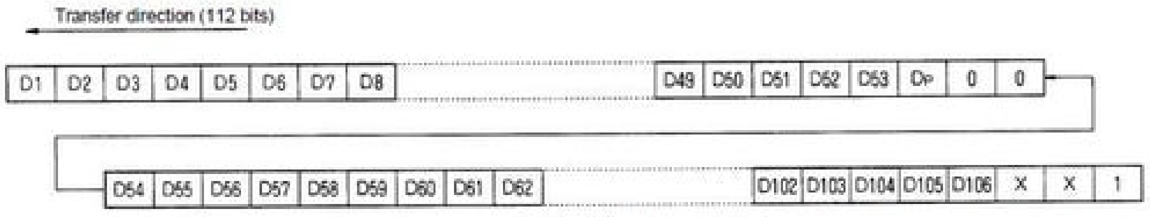


Data Transfer Format

1. Static (1/1) duty

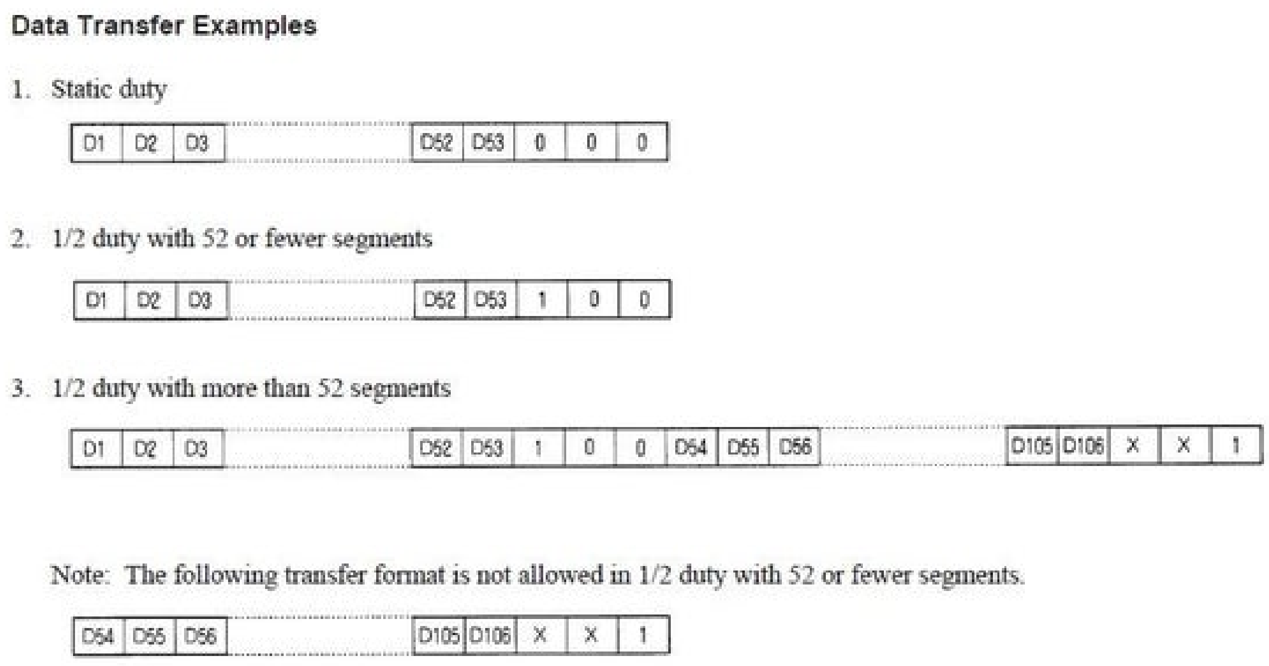


2. 1/2 duty (Only 56 bits need to be transferred if there are no more than 52 display segments. The transfer format is identical to the static duty case. It is not possible to change the D54 to D106 data without specifying the D1 to D53 data.)



The values of bits D53 and D106 are ignored.(don't care)

D1 to D53:	Display data (1/1 duty) Lighted at 1
D1 to D106:	Display data (1/2 duty) Unlighted at 0
DP:	Drive type selection bit 1/2 duty at 1 1/1 duty at 0
X:	don't care



Data Transfer Examples

1. Static duty

D1	D2	D3	....	D52	D53	0	0	0

2. 1/2 duty with 52 or fewer segments

D1	D2	D3	....	D52	D53	1	0	0

3. 1/2 duty with more than 52 segments

D1	D2	D3	....	D52	D53	1	0	0	D54	D55	D56	....	D105	D106	X	X	1

Note: The following transfer format is not allowed in 1/2 duty with 52 or fewer segments.

D54	D55	D56	....	D105	D106	X	X	1

Download

https://www.instructables.com/FCV/1QCH/LAMIIZW3/FCV1QCHLAMIIZW3.pdf

Step 3: Sanyo LC75342 Single-Chip Volume and Tone Control System

The LC75342 and LC75342M are electronic volume and tone control systems that provide volume, balance, a 2-band equalizer, and input switching functions that can be controlled from serially transferred data.

Functions

• Volume: 0 dB to –79 dB (in 1-dB steps) and –°, for a total of 81 settings.

The volume can be controlled independently in the left and right channels to implement a balance function.

• Bass boost: Up to +20 dB in 2-dB steps. Peaking characteristics.

• Treble: ±10 dB in 2-dB steps. Shelving characteristics.

• Selector: One of four sets of left/right inputs can be selected.

• Input gain: The input signal can be boosted by from 0 dB to +30 dB in 2-dB steps.

Features

• On-chip buffer amplifiers minimize the number of external components.

• Fabricated in a silicon gate CMOS process to minimize switching noise from internal switches.

• Built-in analog ground reference voltage generation circuit.

• All controls can be set from serially transferred data. Supports the CCB standard.

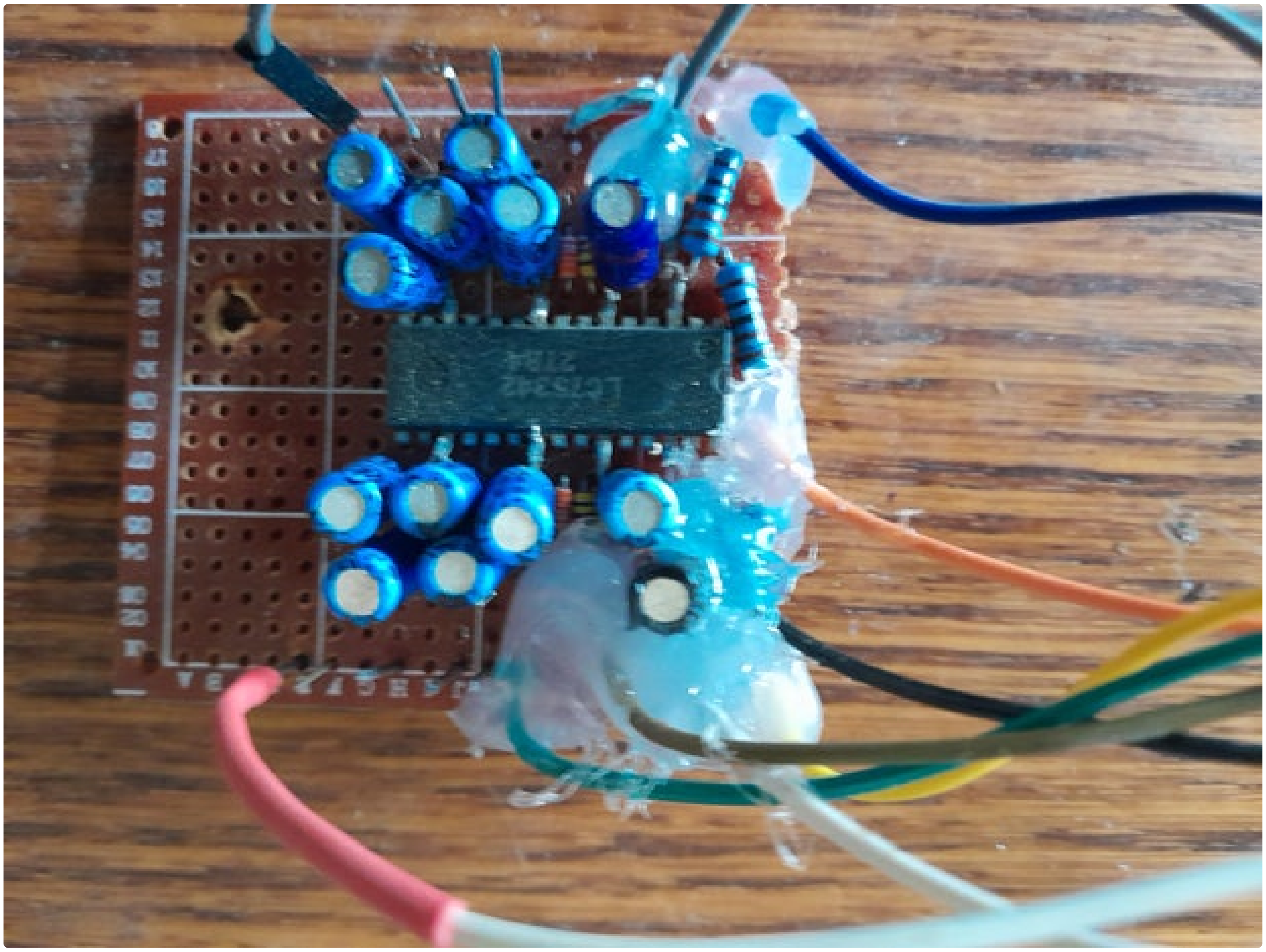
Control System Timing and Data Format

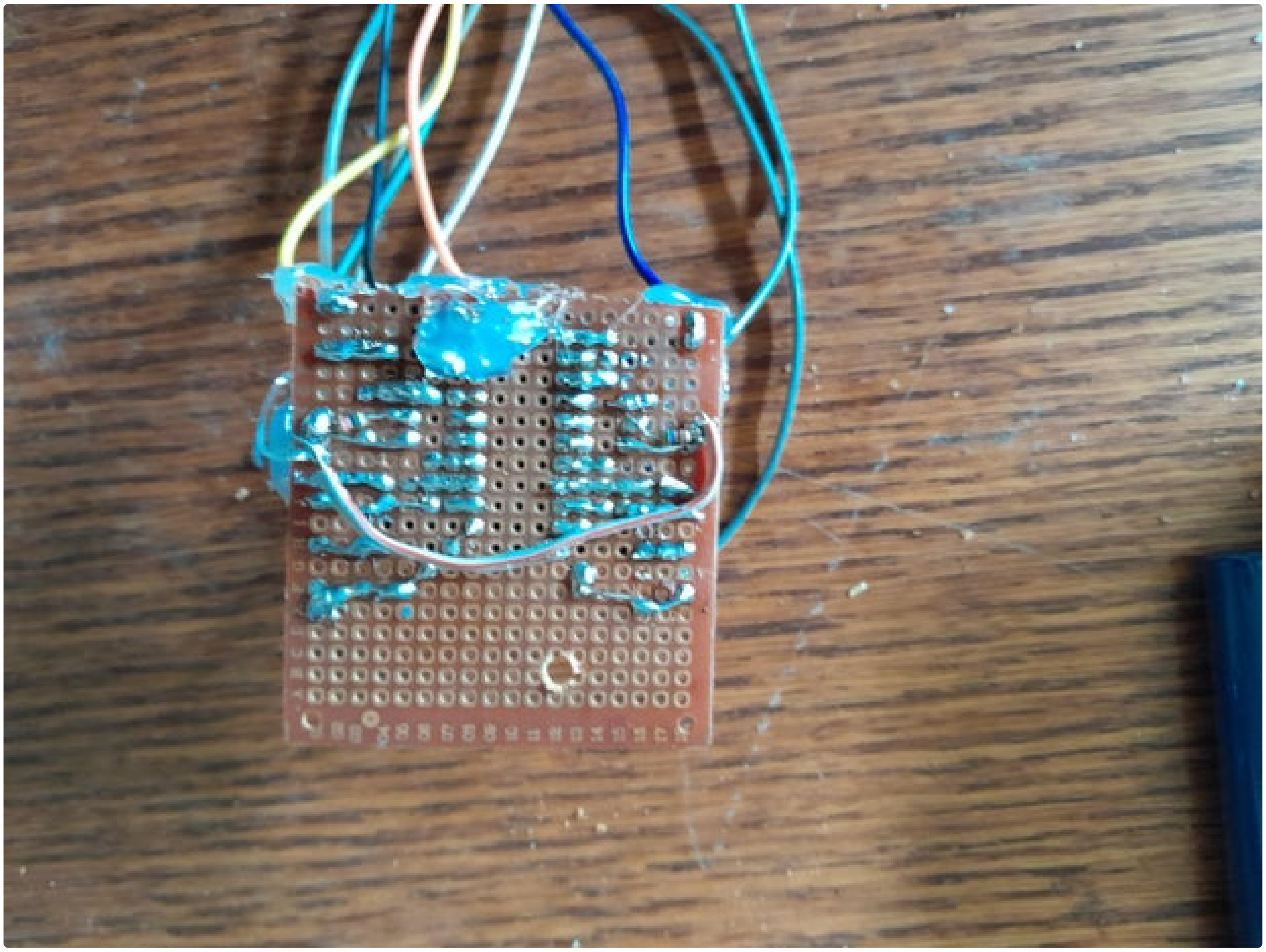
Applications control the LC75342 and LC75342M by applying the stipulated serial data to the CL, DI, and CE pins. This data consists of a total of 40 bits, of which 8 bits are the address and 32 bits are the data itself.

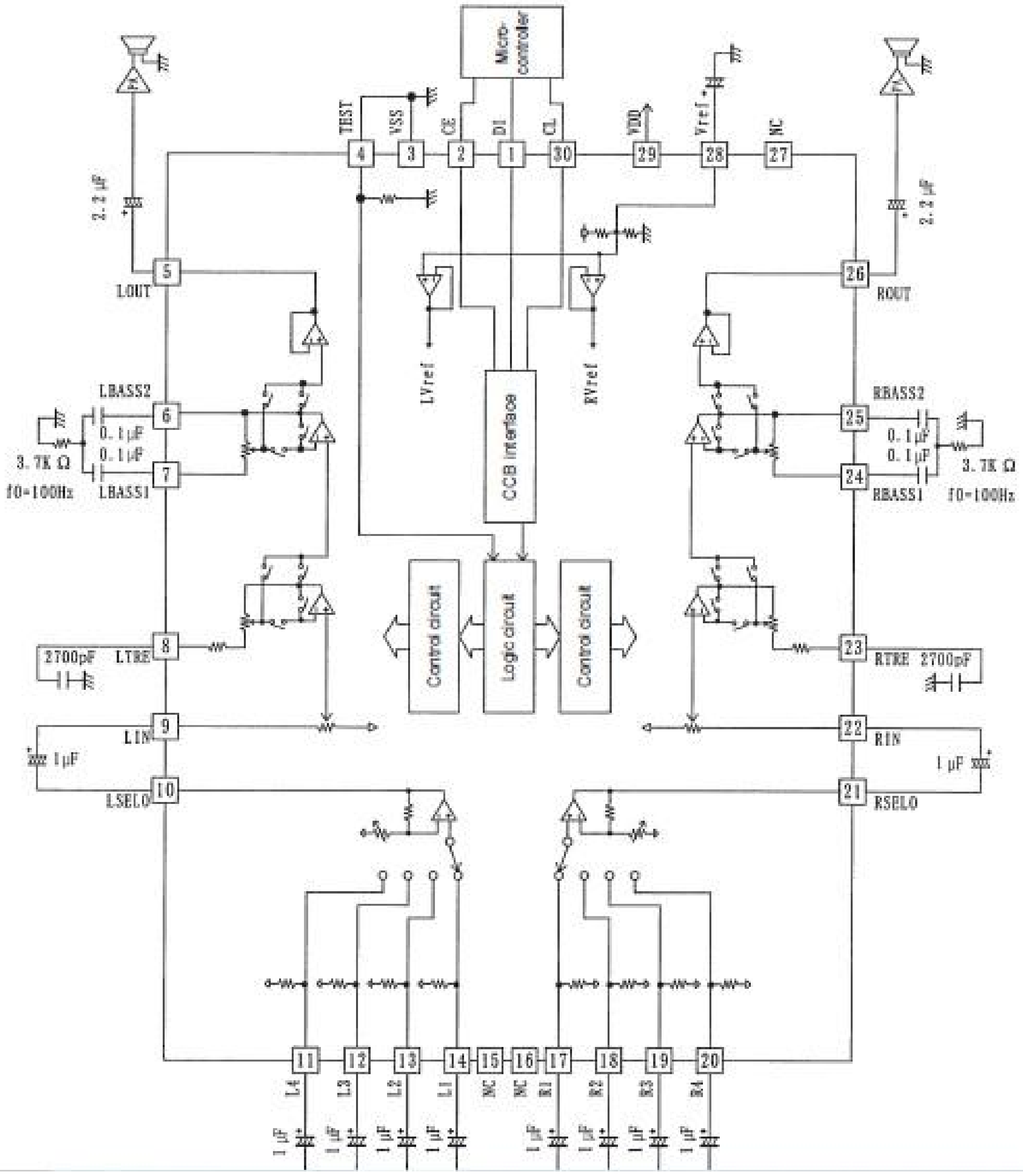
for example **(Control code allocation)** from bit 0 to bit 7 is the IC address and from bit 8 to bit 11 is to select the input switching control, to select like AUX input that connect to bit number 8 =1, and for input gain control from bit number 12 to bit number 15, and for volume control from bit number 13 to bit number 21, and so on for the rest of the function of the IC.

```
#include<SanyoCCB.h>
#include<Arduino.h>
#define LC75342_ADDR  0x82
//Input switching control byte 0
#define LC75_L1_R1 0
#define LC75_L2_R2 1
#define LC75_L3_R3 2
#define LC75_L4_R4 3
#define LC75_IG1   4
#define LC75_IG2   5
#define LC75_IG3   6
#define LC75_IG4   7
  // LC75 volume Control byte1
#define LC75_V0 0
#define LC75_V1 1
#define LC75_V2 2
#define LC75_V3 3
#define LC75_V4 4
#define LC75_V5 5
#define LC75_V6 6
#define LC75_V7 7
 // LC75 treble control and bass control byte2
#define LC75_tc0 0
```

```
#define LC75_tc1 1

#define LC75_tc2 2

#define LC75_tc3 3

#define LC75_bc0 4

#define LC75_bc1 5

#define LC75_bc2 6

#define LC75_bc3 7

//LC75 bass control Continuo and channel selection and test mode byte3

#define LC75_bc4 0

#define LC75_bc5 1

#define LC75_cs0 2

#define LC75_cs1 3

#define LC75_tm0 4

#define LC75_tm1 5

#define LC75_tm2 6

#define LC75_tm3 7

SanyoCCB VolumeAndToneControl(A7, A6, 1, A5); // Pins: DO CL DI CE

byte SignalChipControl[4];

void InputGainControl(byte data){

    bitWrite(SignalChipControl[3], LC75_IG1,  bitRead(data,3));

    bitWrite(SignalChipControl[3], LC75_IG2,  bitRead(data,2));

    bitWrite(SignalChipControl[3], LC75_IG3,  bitRead(data,1));

    bitWrite(SignalChipControl[3], LC75_IG4,  bitRead(data,0));

}

void TrebleControl(byte data){

   bitWrite(SignalChipControl[1],LC75_tc0,  bitRead(data,3));

   bitWrite(SignalChipControl[1],LC75_tc1,  bitRead(data,2));

   bitWrite(SignalChipControl[1],LC75_tc2,  bitRead(data,1));

   bitWrite(SignalChipControl[1],LC75_tc3,  bitRead(data,0));


}

void BaseControl(byte data){

    bitWrite(SignalChipControl[1],LC75_bc0,bitRead(data,5));

     bitWrite(SignalChipControl[1],LC75_bc1,bitRead(data,4));

    bitWrite(SignalChipControl[1],LC75_bc2,bitRead(data,3));

     bitWrite(SignalChipControl[1],LC75_bc3,bitRead(data,2));

    bitWrite(SignalChipControl[0],LC75_bc4,bitRead(data,1));
    bitWrite(SignalChipControl[0],LC75_bc5,bitRead(data,0));


}
void VolumeControl(byte vol){
     bitWrite(SignalChipControl[2],LC75_V0,  bitRead(vol,7));
     bitWrite(SignalChipControl[2],LC75_V1,  bitRead(vol,6));
     bitWrite(SignalChipControl[2],LC75_V2,  bitRead(vol,5));
     bitWrite(SignalChipControl[2],LC75_V3,  bitRead(vol,4));
     bitWrite(SignalChipControl[2],LC75_V4,  bitRead(vol,3));
     bitWrite(SignalChipControl[2],LC75_V5,  bitRead(vol,2));
     bitWrite(SignalChipControl[2],LC75_V6,  bitRead(vol,1));
     bitWrite(SignalChipControl[2],LC75_V7,  bitRead(vol,0));
}
void InputSwitchingControl(byte data){


     bitWrite(SignalChipControl[3], LC75_L1_R1,bitRead(data,3));
     bitWrite(SignalChipControl[3], LC75_L2_R2,bitRead(data,2));
     bitWrite(SignalChipControl[3], LC75_L3_R3,bitRead(data,1));
     bitWrite(SignalChipControl[3], LC75_L4_R4,bitRead(data,0));
     /*
    bitWrite(SignalChipControl[3], LC75_L1_R1,1);
     bitWrite(SignalChipControl[3], LC75_L2_R2,0);
     bitWrite(SignalChipControl[3], LC75_L3_R3,0);
     bitWrite(SignalChipControl[3], LC75_L4_R4,0);
     */
}
void ChannelSelection(byte data){
  bitWrite(SignalChipControl[0],LC75_cs0,bitRead(data, 1));
  bitWrite(SignalChipControl[0],LC75_cs1,bitRead(data, 0));
}
```


LC75342
2784



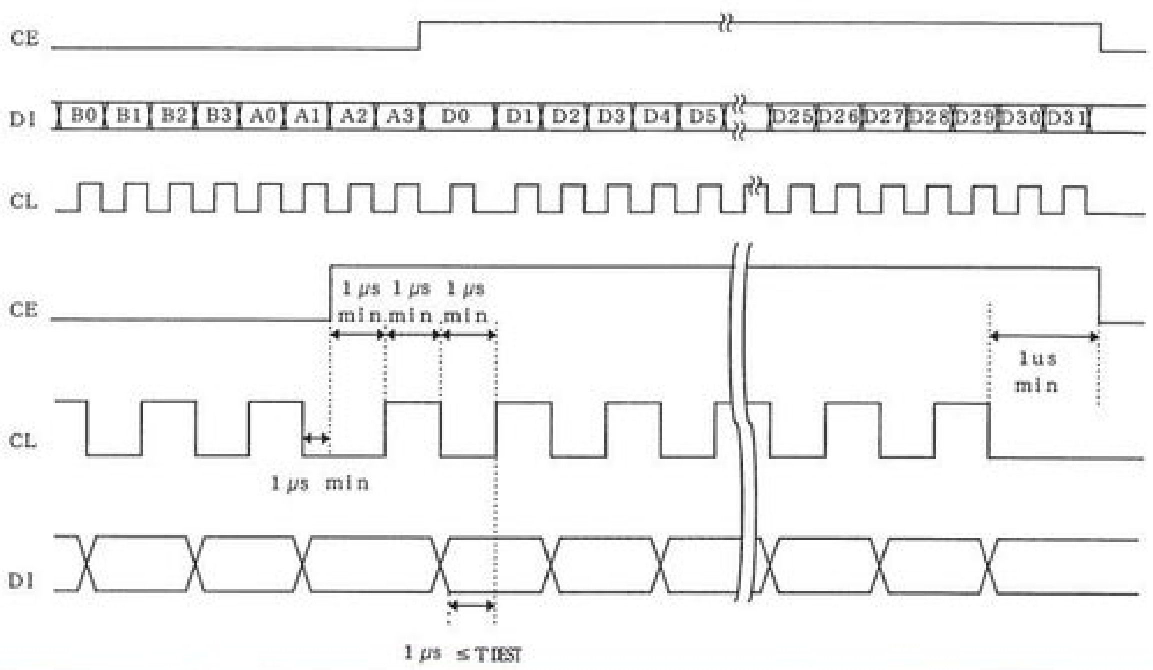

Micro-controller
TEST
VSS
CE
DI
CL
VDD
Vref
NC
2.2 μF
LOUT
ROUT
LVref
RVref
CCB interface
LBASS2
LBASS1
RBASS2
RBASS1
0.1μF
3.7K Ω
f0=100Hz
Control circuit
Logic circuit
2700pF
LTRE
RTRE
LIN
RIN
1μF
LSELO
RSELO
L4
L3
L2
L1
NC
R1
R2
R3
R4

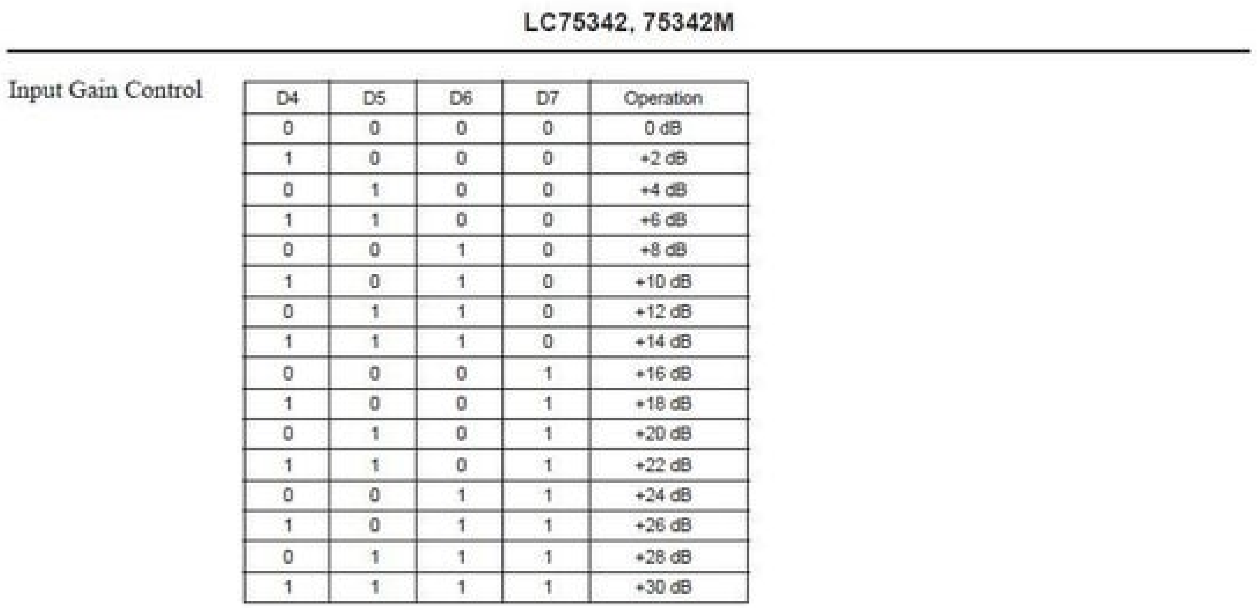
Volume Control

D8	D9	D10	D11	D12	D13	D14	D15	Operation
0	0	0	0	0	0	0	0	0 dB
1	0	0	0	0	0	0	0	–1 dB
0	1	0	0	0	0	0	0	–2 dB
1	1	0	0	0	0	0	0	–3 dB
0	0	1	0	0	0	0	0	–4 dB
1	0	1	0	0	0	0	0	–5 dB
0	1	1	0	0	0	0	0	–6 dB
1	1	1	0	0	0	0	0	–7 dB
0	0	0	1	0	0	0	0	–8 dB
1	0	0	1	0	0	0	0	–9 dB
0	1	0	1	0	0	0	0	–10 dB
1	1	0	1	0	0	0	0	–11 dB
0	0	1	1	0	0	0	0	–12 dB
1	0	1	1	0	0	0	0	–13 dB
0	1	1	1	0	0	0	0	–14 dB
1	1	1	1	0	0	0	0	–15 dB
0	0	0	0	1	0	0	0	–16 dB
1	0	0	0	1	0	0	0	–17 dB
0	1	0	0	1	0	0	0	–18 dB
1	1	0	0	1	0	0	0	–19 dB
0	0	1	0	1	0	0	0	–20 dB
1	0	1	0	1	0	0	0	–21 dB
0	1	1	0	1	0	0	0	–22 dB
1	1	1	0	1	0	0	0	–23 dB
0	0	0	1	1	0	0	0	–24 dB
1	0	0	1	1	0	0	0	–25 dB

Control System Timing and Data Format

Applications control the LC75342 and LC75342M by applying the stipulated serial data to the CL, DI, and CE pins. This data consists of a total of 40 bits, of which 8 bits are the address and 32 bits are the data itself.

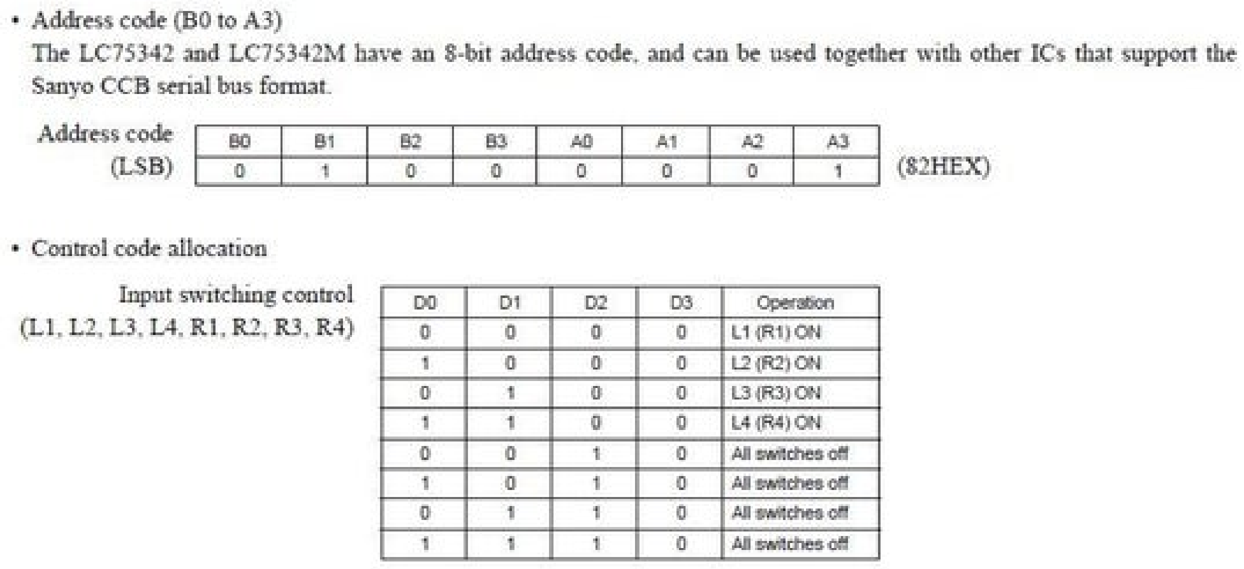




LC75342, 75342M

Input Gain Control

D4	D5	D6	D7	Operation
0	0	0	0	0 dB
1	0	0	0	+2 dB
0	1	0	0	+4 dB
1	1	0	0	+6 dB
0	0	1	0	+8 dB
1	0	1	0	+10 dB
0	1	1	0	+12 dB
1	1	1	0	+14 dB
0	0	0	1	+16 dB
1	0	0	1	+18 dB
0	1	0	1	+20 dB
1	1	0	1	+22 dB
0	0	1	1	+24 dB
1	0	1	1	+26 dB
0	1	1	1	+28 dB
1	1	1	1	+30 dB



- Address code (B0 to A3)
  The LC75342 and LC75342M have an 8-bit address code, and can be used together with other ICs that support the Sanyo CCB serial bus format.

Address code (LSB)

B0	B1	B2	B3	A0	A1	A2	A3	
0	1	0	0	0	0	0	1	(82HEX)

- Control code allocation

Input switching control (L1, L2, L3, L4, R1, R2, R3, R4)

D0	D1	D2	D3	Operation
0	0	0	0	L1 (R1) ON
1	0	0	0	L2 (R2) ON
0	1	0	0	L3 (R3) ON
1	1	0	0	L4 (R4) ON
0	0	1	0	All switches off
1	0	1	0	All switches off
0	1	1	0	All switches off
1	1	1	0	All switches off

Download

https://www.instructables.com/FEY/0U9L/LAMIIZW1/FEY0U9LLAMIIZW1.pdf

Step 4: TPA3101D2 10-W STEREO CLASS-D AUDIO POWER AMPLIFIER

.I did cut this board from an old LCD TV it is an Class-D audio power amplifier, I did read the data sheet first before I cut it from the TV to mark all the rea for the IC connection .

FEATURES

10-W/ch into an 8-Ω Load From a 13-V Supply • Televisions

• 9.2-W/ch into an 8-Ω Load From a 12-V Supply

DESCRIPTION • Operates from 10 V to 26 V

• 87% Efficient Class-D Operation Eliminates The TPA3101D2 is a 10-W (per channel) efficient,

Need for Heat Sinks Class-D audio power amplifier for driving bridged-tied

stereo speakers. The TPA3101D2 can drive stereo

• Four Selectable, Fixed Gain Settings speakers as low as 4 Ω. The high efficiency of the

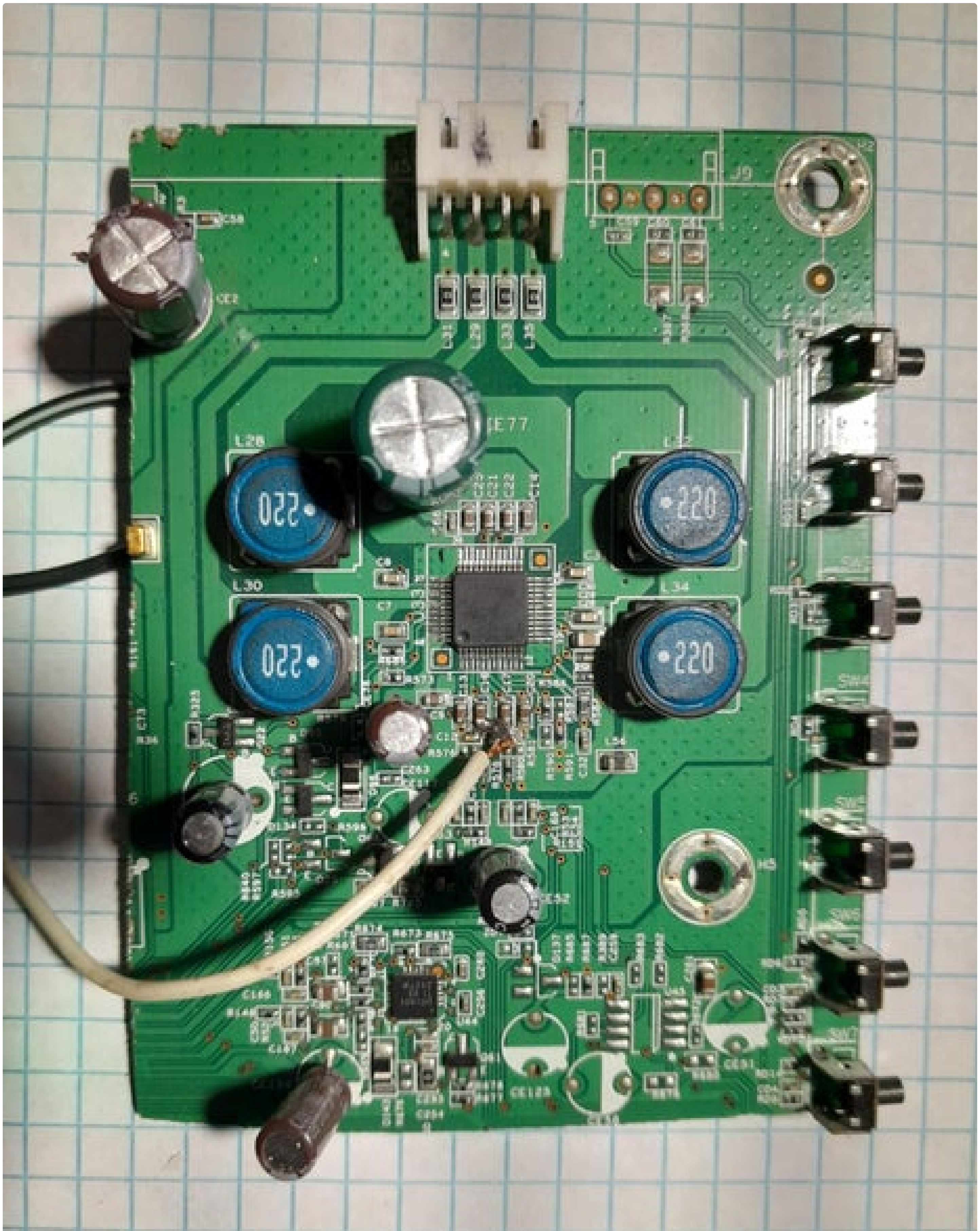
• Differential Inputs TPA3101D2, 87%, eliminates the need for an

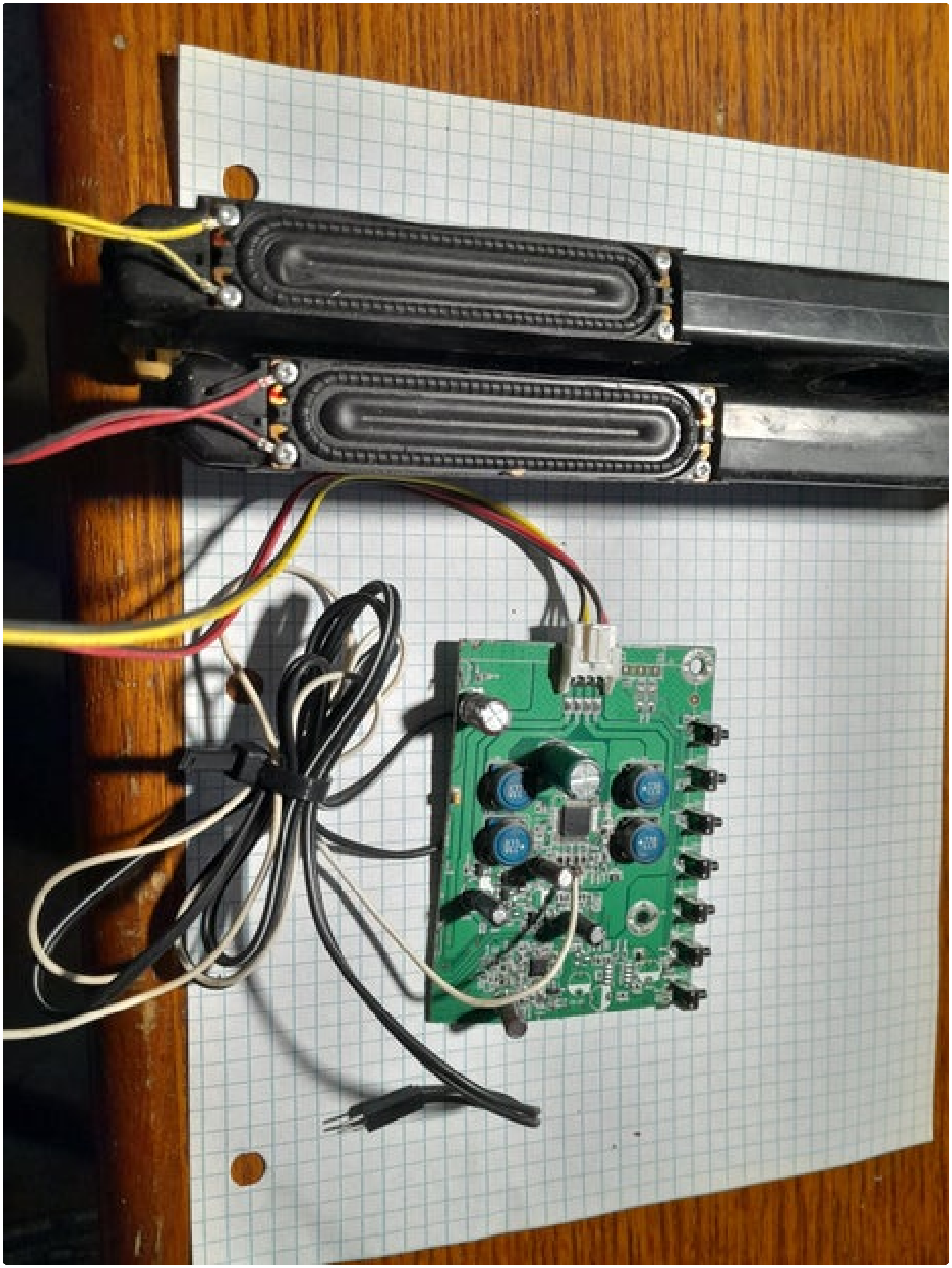
• Thermal and Short-Circuit Protection With external heat sink when playing music.

Auto Recovery Feature The gain of the amplifier is controlled by two gain

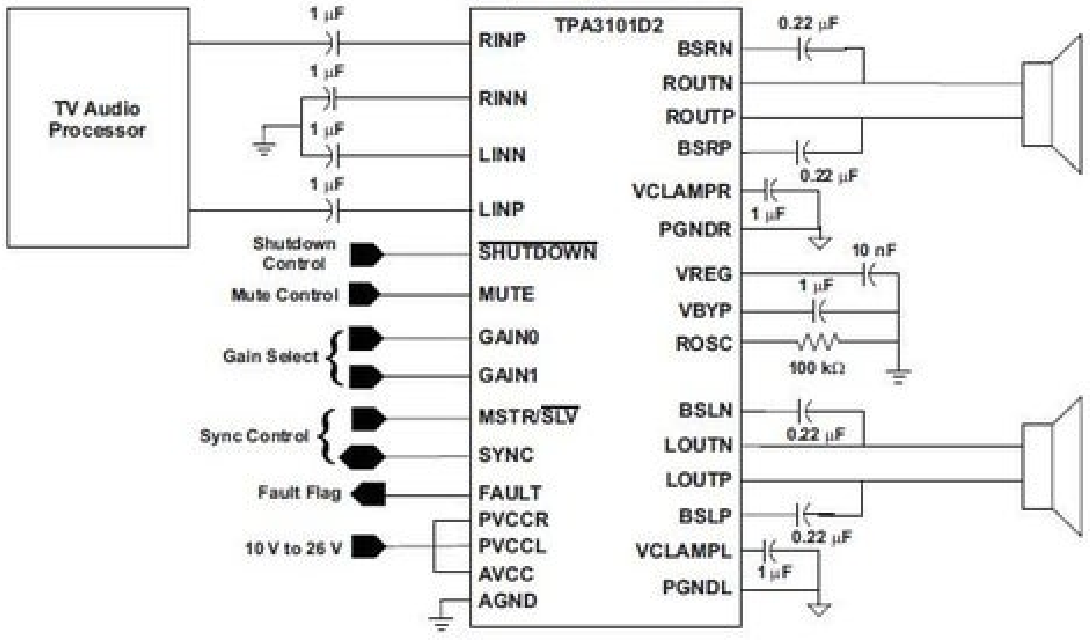
• Clock Output for Synchronization With select pins. The gain selections are 20, 26, 32,

Multiple Class-D Devices 36 dB.

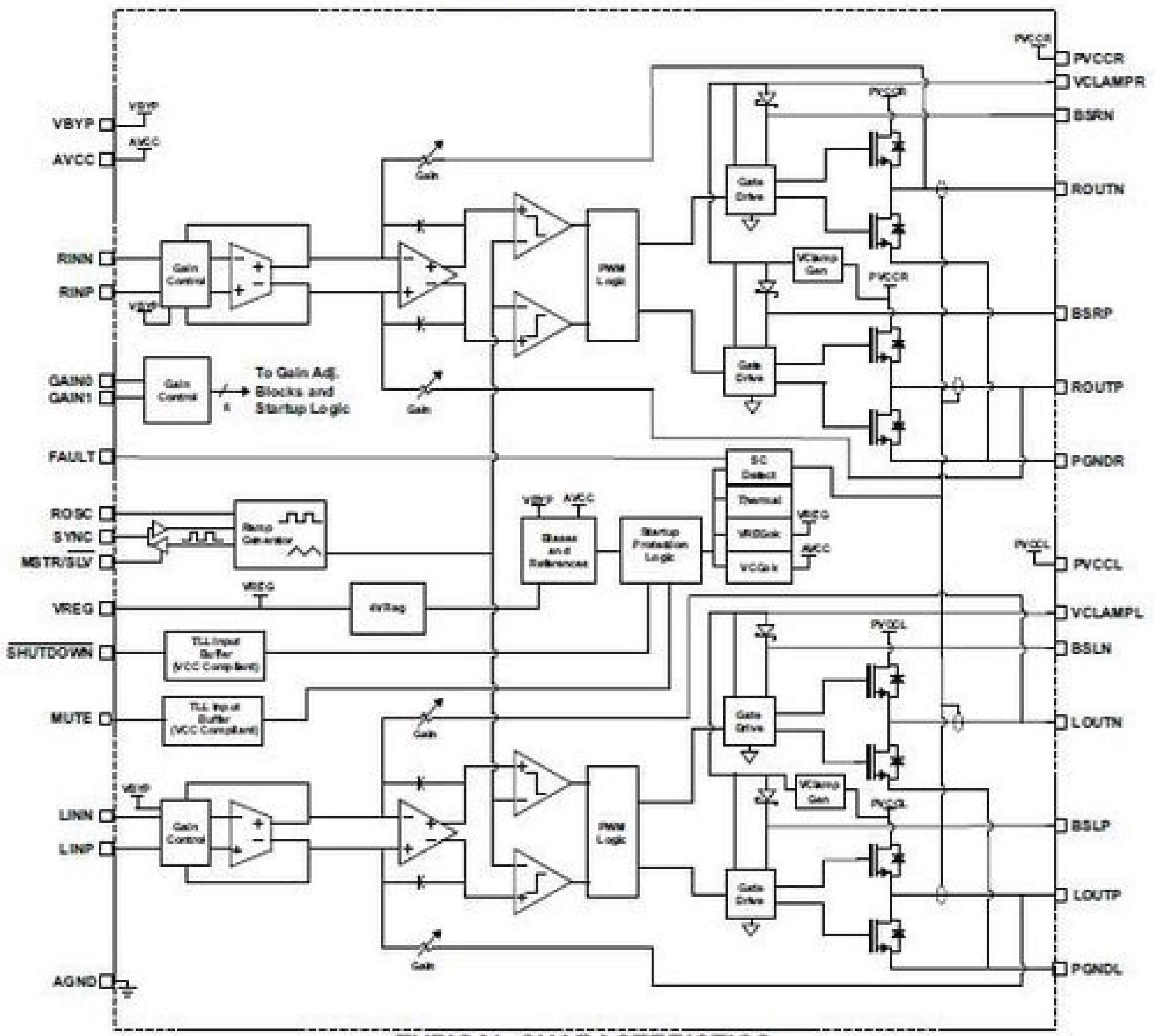




Simplified Application Circuit



FUNCTIONAL BLOCK DIAGRAM



TYPICAL CHARACTERISTICS

https://www.instructables.com/FSI/TH2D/LAMIJ996/FSITH2DLAMIJ996.pdf Download

Step 5: Remote Control IR Reciver

First I did use IRremote library to decode each button in the remote control, and use that value for specific function

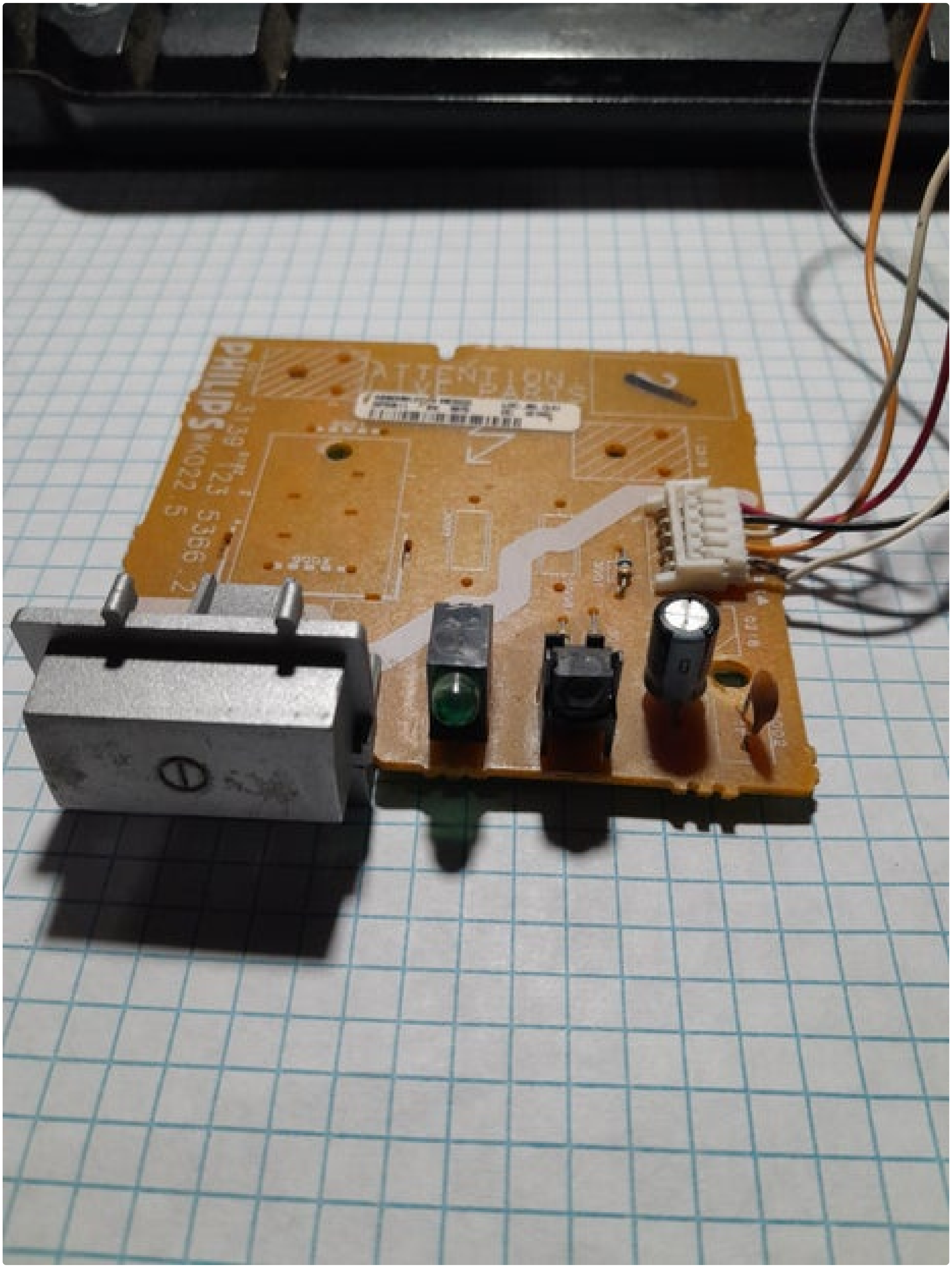
```
#include <Arduino.h>
#include <IRremote.h
int RECV_PIN = 2;  // IR Receiver pin
IRrecv irrecv(RECV_PIN);
decode_results results;
void setup() {
 // put your setup code here, to run once:
 irrecv.enableIRIn();
 Serial.begin(9600);
 Serial.println(results.value,DEC);
}

void loop() {
 // put your main code here, to run repeatedly:
 if (irrecv.decode(&results)){
     Serial.println(results.value,DEC);
    irrecv.resume();
}
```

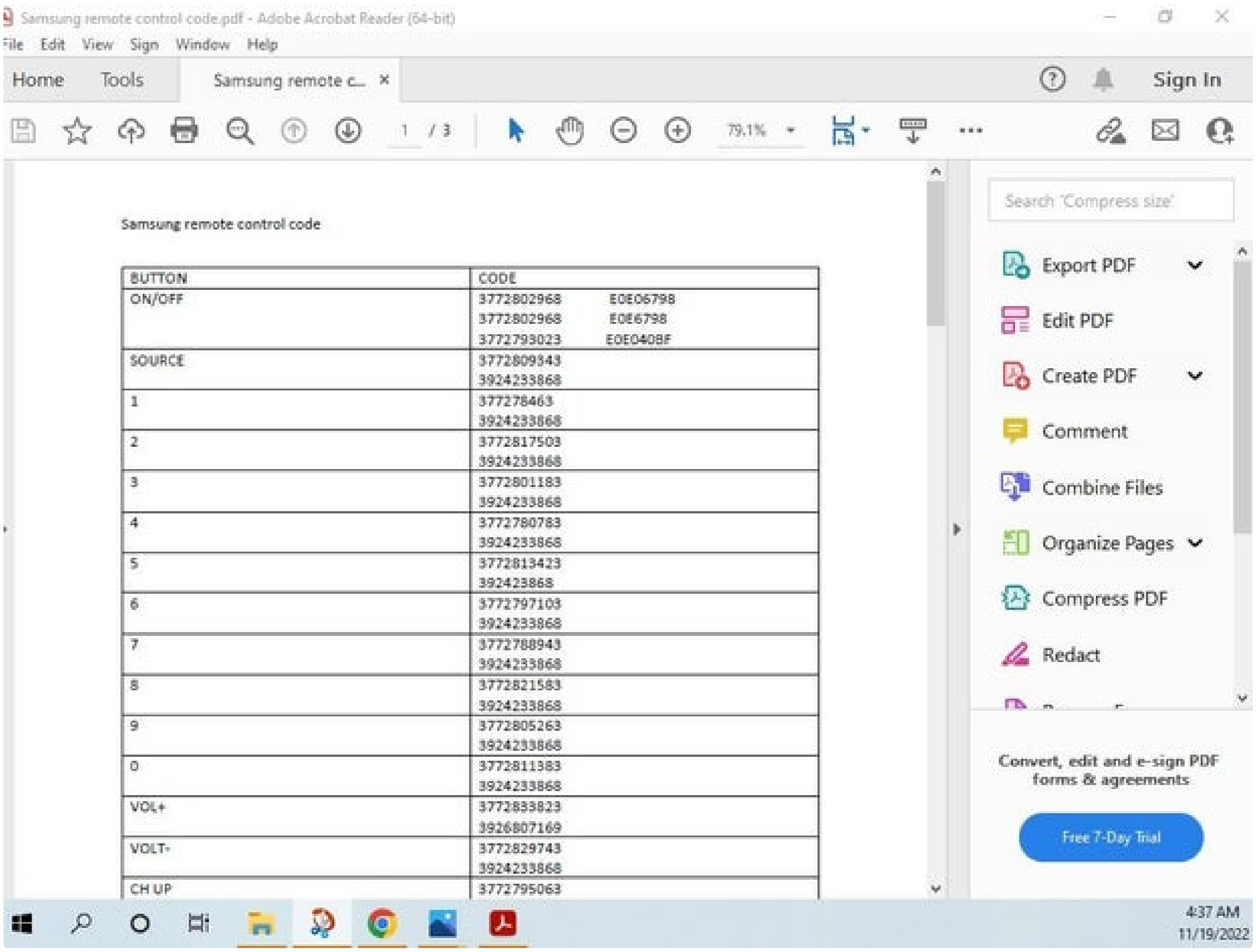
the use for the results.value

```
#include<SanyoLCD.h>
#include<Single-Chip Volume and Tone Control System.h>
#include<SanyoRadioPLL.h>
#include <IRremote.h>
#include<LiquidCrystal_I2C.h>
#define clk 6
#define dt 5
int cuk=0
int pvk=0;
//FMFrequency = 949;
LiquidCrystal_I2C  lcd(0x27, 16, 2);
int RECV_PIN = 2;  // IR Receiver pin
```

```
IRrecv irrecv(RECV_PIN);
decode_results results;
int vol=48;
void setup() {
 lcd.init();
  lcd.begin(16,2);
 lcd.backlight();
irrecv.enableIRIn();
 LCDDisplay.init();
 VolumeAndToneControl.init();
 InputSwitchingControl(1,vol);
 Serial.begin(9600);
 pinMode(clk,INPUT);
 pinMode(dt,INPUT);
 pinMode(INH,OUTPUT);
 Radioccb.init();
 LC72131Init();
 band = LC72131_BAND_FM;
LC72131SetMode(LC72131_BAND_FM);
 tuned = LC72131Tune(FMFrequency);
 Display(FMFrequency);

  lcd.setCursor(0,0);
  lcd.print("VOL");
 lcd.setCursor(4,0);
 lcd.print(vol);
pvk = digitalRead(clk);
}



void loop() {
 if (irrecv.decode(&results)){
   switch (results.value)
   {
     case 3772780783: //AUX
     InputSwitchingControl(4,vol);
   lcd.setCursor(0,0);
  lcd.print("VOL");
  lcd.setCursor(4,0);
  lcd.print(vol);
  Serial.println(vol);
  break;
case 3772795063 : //FM radio channel up
   FMFrequency++;
   // AMFrequency++;
 LC72131Tune(FMFrequency);
  if(FMFrequency > 1080) FMFrequency = 1080;
   Display(FMFrequency);
  break;
case 3772778743 : // FM radio  channel down
 FMFrequency--;
  // AMFrequency--;
LC72131Tune(FMFrequency);
 //if(AMFrequency < 53)  AMFrequency = 170;
 if(FMFrequency < 880)  FMFrequency = 880;
 Display(FMFrequency);
break;
 case 3772833823://volum UP
  vol++;
  InputSwitchingControl(1,vol);
  lcd.setCursor(0,0);
  lcd.print("VOL");
  lcd.setCursor(4,0);
  lcd.print(vol);
  Serial.println(vol);
    if(vol >= 80)vol=80;
  break;
   case 3772829743://Volum DOWN
 vol--;
 InputSwitchingControl(1,vol);
  lcd.setCursor(0,0);
  lcd.print("VOL");
  lcd.setCursor(4,0);
```

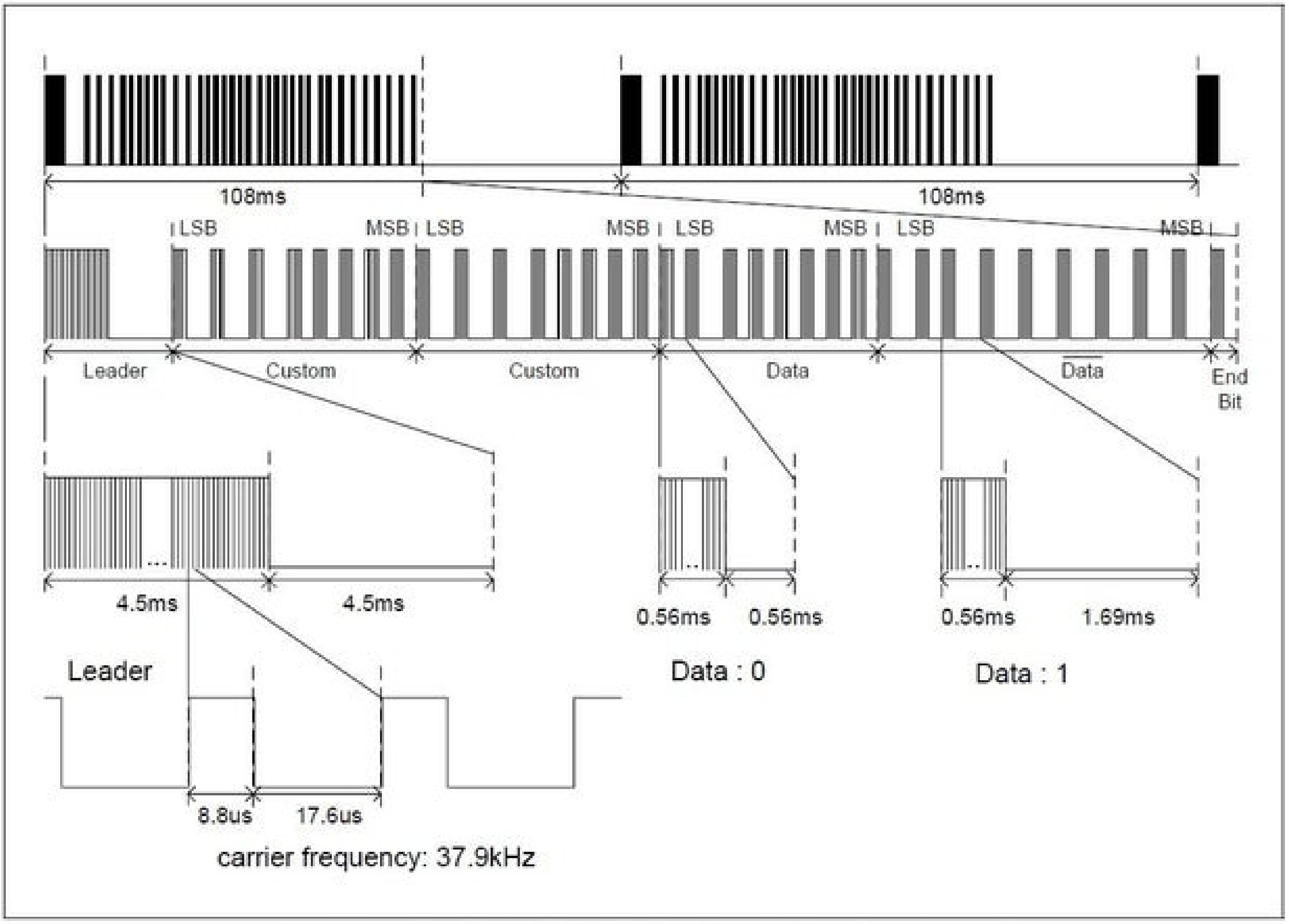
```
      lcd.print(vol);
     Serial.println(vol);
     if(vol <=0)vol=1;
    break;
    }
   irrecv.resume();
  cuk = digitalRead(clk);
  if(cuk != pvk){
   if(cuk != digitalRead(dt)){
     FMFrequency--;
      // AMFrequency--;
     LC72131Tune(FMFrequency);
    if(FMFrequency < 880)  FMFrequency = 880;
     Display(FMFrequency);
    Serial.print("FM ");
    Serial.println((float)FMFrequency / 10,1);
   }
   else{
     FMFrequency++;
  // AMFrequency++;
    LC72131Tune(FMFrequency);
      if(FMFrequency > 1080) FMFrequency = 1080;
       Display(FMFrequency);
       Serial.print("FM ");
       Serial.println((float)FMFrequency / 10,1);
   }
  }
 pvk = cuk;

}
```

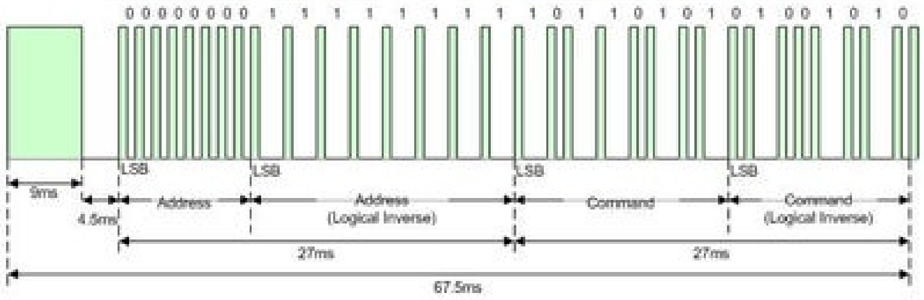

PHILIPS
3139 123 5366 2
WK022.5
ATTENTION



Samsung remote control code

BUTTON	CODE
ON/OFF	3772802968 E0E06798 3772802968 E0E6798 3772793023 E0E040BF
SOURCE	3772809343 3924233868
1	377278463 3924233868
2	3772817503 3924233868
3	3772801183 3924233868
4	3772780783 3924233868
5	3772813423 392423868
6	3772797103 3924233868
7	3772788943 3924233868
8	3772821583 3924233868
9	3772805263 3924233868
0	3772811383 3924233868
VOL+	3772833823 3926807169
VOLT-	3772829743 3924233868
CH UP	3772795063





Step 6: The Use of This Project

I did create source file for each IC and I reference that source file in main.

https://www.youtube.com/watch?v=BMuHiksRTD4

Download

https://www.instructables.com/FNK/78XS/LAPDIE3B/FNK78XSLAPDIE3B.h

Download

https://www.instructables.com/FB3/052D/LAPDIE3A/FB3052DLAPDIE3A.h

https://www.instructables.com/F6I/TMC4/LAPDIE39/F6ITMC4LAPDIE39.h Download

https://www.instructables.com/F2G/OMNY/LAPDIE38/F2GOMNYLAPDIE38.cpp Download