Samsung A-C Smart Controller

In this page we will explain how the reverse engineering for the Samsung Air Conditioner Infrared Remote Controller was made, how the parameters were acquired, and how to build the on, off, and command (temperature, swing, fan speed, mode, and turbo) messages.

Supported Air Conditioners

For the reverse engineering process, we used a Infrared Remote Controller that comes with the Samsung AC models:

- Cold Models: ASV09P Series\ASV12P Series\ASV18P Series\ASV24P Series\AR09HV Series\AR12HV Series\AR18HV Series\AR24HV Series.
- Reverse Models (Hot/Cold): AQV09P Series\AQV12P Series\AQV18P Series\AQV24P Series\AR09HS Series\AR12HS Series\AR18HS Series\AR24HS Series.

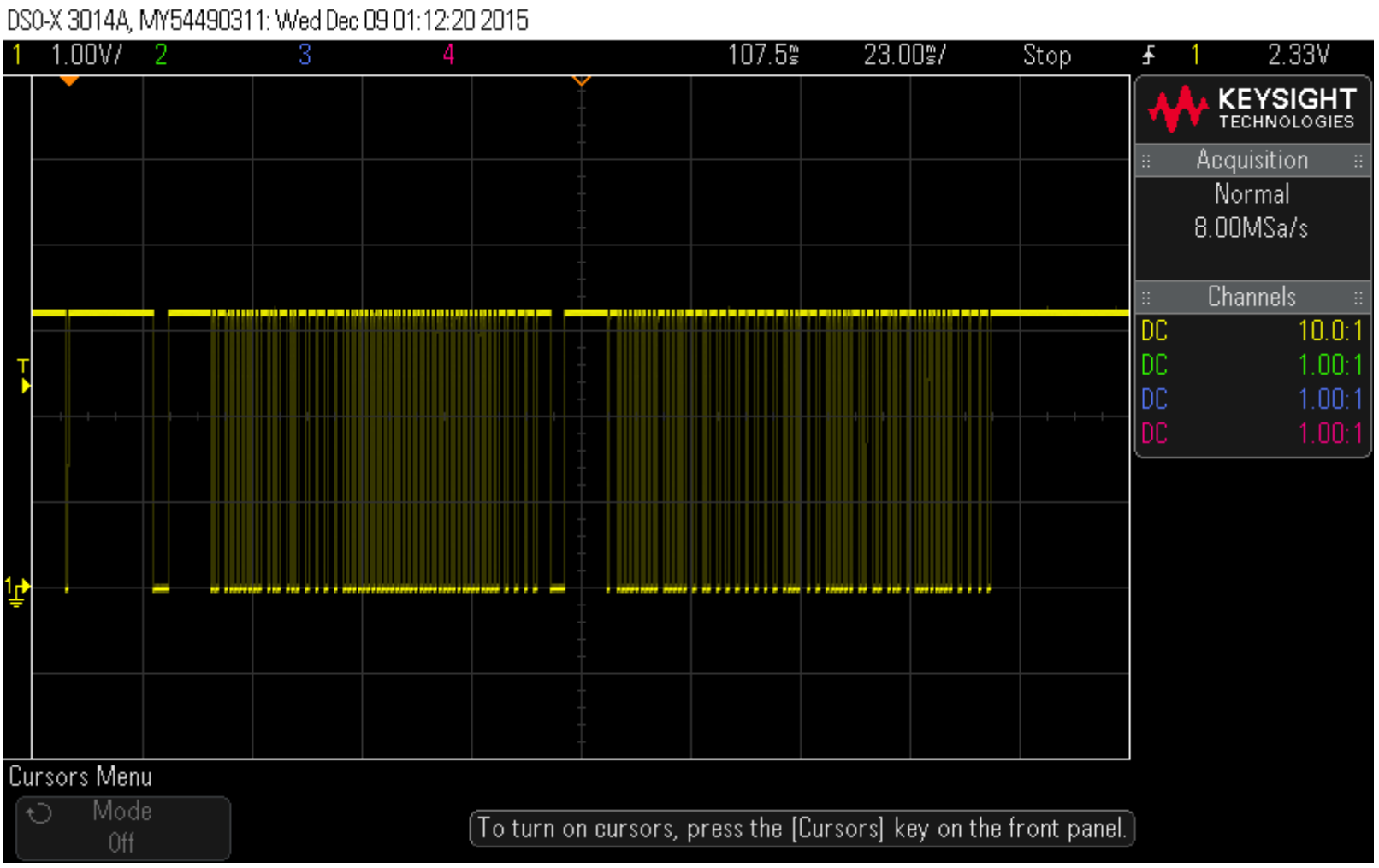
The google search of information about the codes for Samsung AC IR Controller returned two types of results: an Arduino implementation with raw data for generic IR Controllers 1 and some information about the the Samsung protocol for TVs 2.

The Reverse Engineering Process

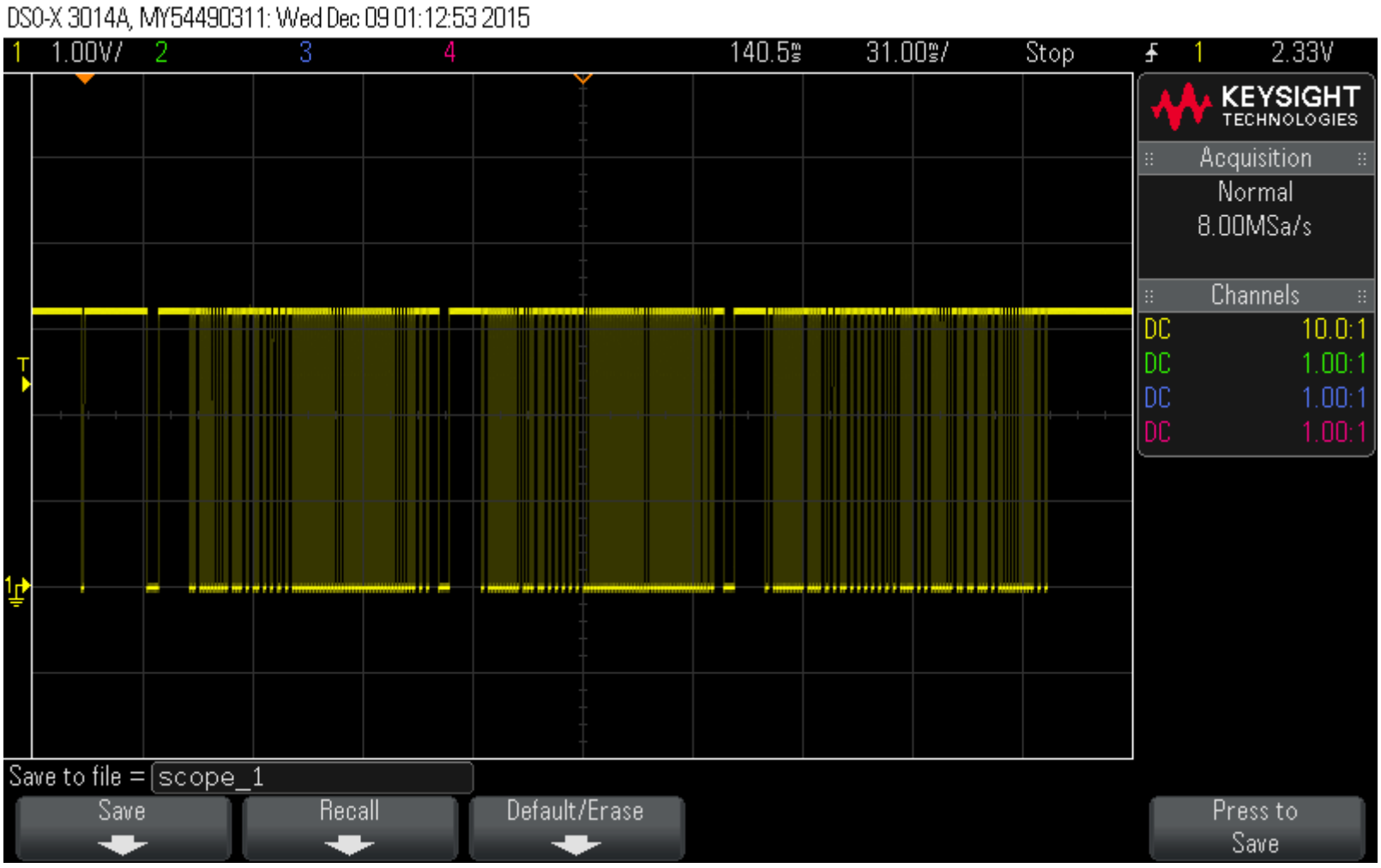
The first critical information was that data sent by an AC IR Controller is more elaborate than data sent by a TV IR Controller. The data processing for the AC IR Controller is made in the Controller, all the variables are sent in one message: the temperature, mode, fan speed, swing, and turbo. For TV Controllers, just the pressed button information is sent.

There is a paid software for decoding the RAW data from the IR Controller 3, the software only analyses the raw data coming from Arduino. Using this process as example, we used the Texas Instrument BOOST-IR Boosterpack 4 with a TI Tiva Laucnhpad 5 to get the raw data. Initially, the only function of the Tiva Launchpad was supply the power for the Boosterpack.

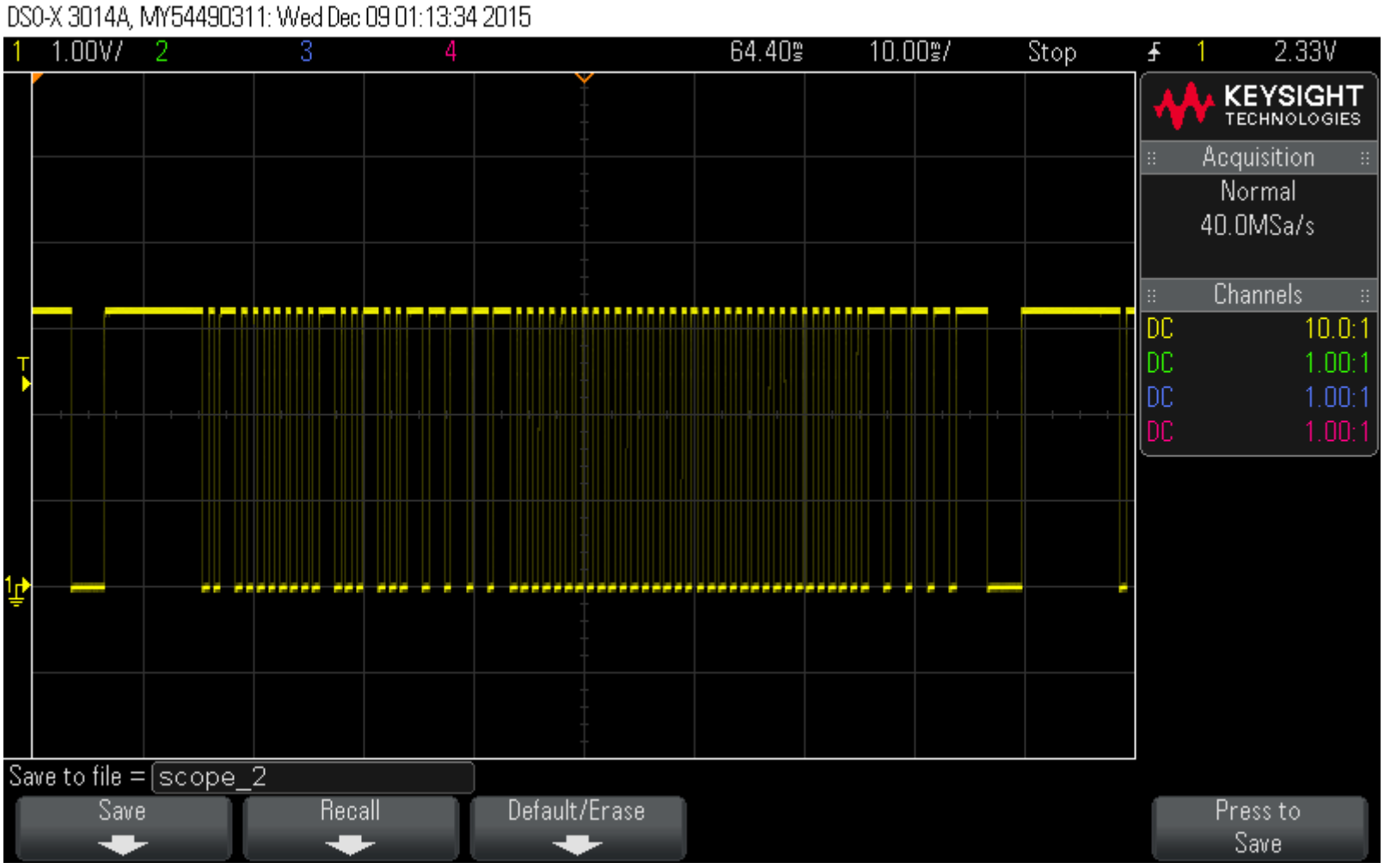
The result for a command sent by the Samsumg AC IR Controller is shown below. A significant feature is that we see two blocks of data, this occurs when a configuration command is sent. When a turn on/off signal is sent, we have three data blocks. Other aspect is the header, when the message starts we have always the same sequence of high/low times, same as the "interblock" interval.



Data sent by the Samsung IR Controller when temperature button is pressed (command message).

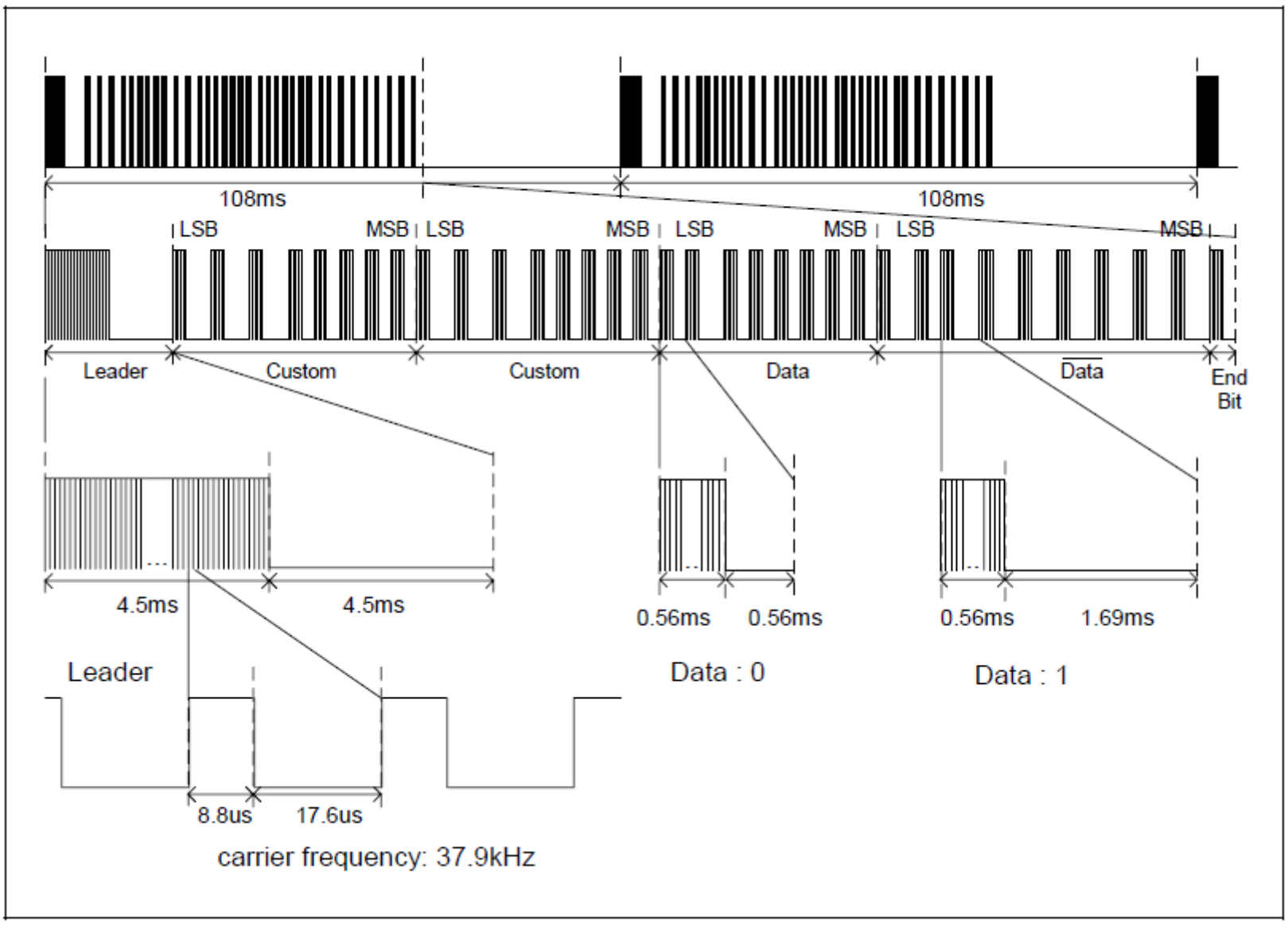


Data sent by the Samsung IR Controller when ON/OFF button is pressed (ON/OFF message).



First data block in detail in the ON/OFF message.

The Samsumg Application Note 1 explains how to decode the signal for '0' and '1'.



How to decode the signal for zero and one.

Using the patterns for '0' and '1', we see that the first data block is always the same for command messages (with the value 0x4049F00000000F). The second data block changes when we press diferent buttons, not all the commands were tested, but the ones that we have tested are shown below.

Sample	Mode	Temperature	Fan Speed	Turbo	Swing	Data value
1	Cool	16	2	OFF	OFF	0x80477F8E00980F
2	Cool	17	2	OFF	OFF	0x804B7F8E08980F
3	Cool	18	2	OFF	OFF	0x804B7F8E04980F
4	Cool	19	2	OFF	OFF	0x80437F8E0C980F
5	Cool	20	2	OFF	OFF	0x804B7F8E02980F
6	Cool	21	2	OFF	OFF	0x80437F8E06980F
7	Cool	28	2	OFF	OFF	0x80437F8E03980F
8	Cool	29	2	OFF	OFF	0x804D7F8E0B980F
9	Cool	30	2	OFF	OFF	0x804D7F8E07980F
10	Cool	20	Auto	OFF	OFF	0x80477F8E02880F
11	Cool	20	Auto	ON	OFF	0x80437FEE02880F
12	Cool	20	Auto	OFF	ON	0x8040F58E02880F
13	Dry	18	Auto	OFF	OFF	0x80477F8E04840F
14	Fan	Not Avaible	1	OFF	OFF	0x80437F8E01AC0F
15	Fan	Not Avaible	2	OFF	OFF	0x80437F8E019C0F
16	Fan	Not Avaible	3	OFF	OFF	0x804D7F8E01DC0F
17	Heat	30	Auto	OFF	OFF	0x80437F8E07820F
18	Heat	30	3	OFF	OFF	0x80457F8E07D20F
19	Auto	20	Auto	OFF	OFF	0x804B7F8E02B00F

In order to determine how to build the command message, we observed the differences between two messages when changing only one parameter. We present the results in the following table:

Sample	1st Byte	2nd Byte	3rd Byte	4th Byte	5th Byte	6th Byte	7th Byte	Description
1	1000 0000	0100 0111	0111 1111	1000 1110	0000 0000	1000 1000	0000 1111	Cool 16 Fan 2 T OFF S OFF
2	1000 0000	0100 1011	0111 1111	1000 1110	0000 1000	1001 1000	0000 1111	Cool 17 Fan 2 T OFF S OFF
3	1000 0000	0100 1011	0111 1111	1000 1110	0000 0100	1001 1000	0000 1111	Cool 18 Fan 2 T OFF S OFF
4	1000 0000	0100 0011	0111 1111	1000 1110	0000 1100	1001 1000	0000 1111	Cool 19 Fan 2 T OFF S OFF
5	1000 0000	0100 1011	0111 1111	1000 1110	0000 0010	1001 1000	0000 1111	Cool 20Fan 2 T OFF S OFF
6	1000 0000	0100 0011	0111 1111	1000 1110	0000 0110	1001 1000	0000 1111	Cool 21 Fan 2 T OFF S OFF
7	1000 0000	0100 0011	0111 1111	1000 1110	0000 0011	1001 1000	0000 1111	Cool 28 Fan 2 T OFF S OFF
8	1000 0000	0100 1101	0111 1111	1000 1110	0000 1011	1001 1000	0000 1111	Cool 29 Fan 2 T OFF S OFF
9	1000 0000	0100 1101	0111 1111	1000 1110	0000 0111	1001 1000	0000 1111	Cool 30 Fan 2 T OFF S OFF
10	1000 0000	0100 0111	0111 1111	1000 1110	0000 0010	10001000	0000 1111	Cool 20 Fan AutoT OFFS OFF
11	1000 0000	0100 0011	0111 1111	1110 1110	0000 0010	1000 1000	0000 1111	Cool 20 Fan Auto T ON S OFF
12	1000 0000	0100 0000	1111 0101	1000 1110	0000 0010	1000 1000	0000 1111	Cool 20 Fan Auto T OFF S ON
13	1000 0000	0100 0111	0111 1111	1000 1110	0000 0100	1000 0100	0000 1111	Dry 18 Fan Auto T OFF S OFF
14	1000 0000	0100 0011	0111 1111	1000 1110	0000 0001	10101100	0000 1111	Fan N/A Fan 1 T OFF S OFF
15	1000 0000	0100 0011	0111 1111	1000 1110	0000 0001	1001 1100	0000 1111	Fan N/A Fan 2 T OFF S OFF
16	1000 0000	0100 1101	0111 1111	1000 1110	0000 0001	1101 1100	0000 1111	Fan N/A Fan 3 T OFF S OFF
17	1000 0000	0100 0011	0111 1111	1000 1110	0000 0111	1000 0010	0000 1111	Heat 30 Fan Auto T OFF S OFF
18	1000 0000	0100 0101	0111 1111	1000 1110	0000 0111	1101 0010	0000 1111	Heat 30 Fan 3 T OFF S OFF
19	1000 0000	0100 1011	0111 1111	1000 1110	0000 0010	10110000	0000 1111	Auto 20 Fan Auto T OFF S OFF

We observe these correlations, a set of summed settings is shown below:

- Temperature (red): in byte 5, the bits 0-3 determine the temperature in the modes Cool/Dry/Heat/Auto. The most significant bit (MSb) is the bit0 and the least significant bit (LSb) is the bit3. The temperature of 16ºC is equal to 0 in binary, and 30º is equal to 14 in binary.
- Mode of operation (cian): in byte 6, the bits 3-1 determine the mode of operation of the A-C.
- Fan Speed (green): in byte 6, the bits 6-4 determine the Fan Speed of the A-C. We observed that auto mode have a unique value for Fan Speed Auto, this value is different for the Cool/Dry/Heat/Fan modes.
- Turbo (blue): in byte 4, the bits 6-5 determine whether the Turbo is on or off.
- Swing (yellow): the byte 3 determines if the Swing is on or off.

Temperature	Byte 5 bits3-0
16	0000
17	1000
18	0100
19	1100
20	0010
21	1010
22	0110
23	1110
24	0001
25	1001
26	0101
27	1101
28	0011
29	1011
30	0111

Mode of operation	Byte 6 bits3-1
Cool	100
Dry	010

Fan	110
Heat	001
Auto	000

Fan Speed	Byte 6 bits 6-4	Modes acepted
Auto	000	Cool/Fan/Heat/Dry
1	010	Cool/Fan/Heat
2	001	Cool/Fan/Heat
3	101	Cool/Fan/Heat
Auto	011	Fan

Turbo	Byte 4 bits 6-5
OFF	00
ON	11

Swing	Byte 3
OFF	0111 1111
ON	1111 0101

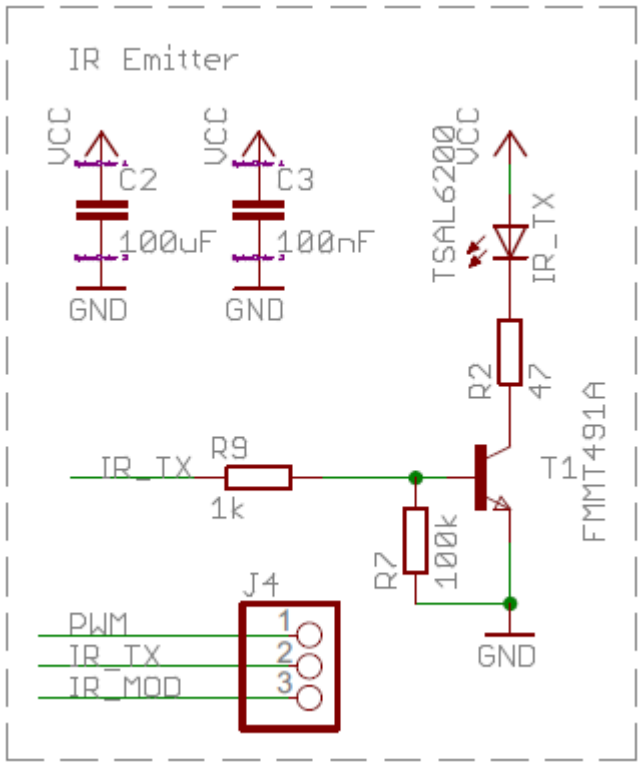
Error Detection Word Calculation

In byte 2, the bits 3-0 are a kind of Error Detection Code, we deduce that the value of this code is proportional to the number of '1' in the bytes 3-6. The code is mirrored, we can se this in the aditional columns of the table below.

Number of '1'	Byte 2 bits 3-0	Mirrored binary	Mirrored Decimal
10	1100	0011	3
11	0100	0010	2
12	1000	0001	1
13	0000	0000	0
14	0111	1110	E
15	1011	1101	D
16	0011	1100	C
17	1101	1011	B
18	0101	1010	A
19	1001	1001	9
20	0001	1000	8
21	1110	0111	7

Message Format

We can send the message through the use of two types of hardware connections. In this Section we show how to build the message for Infrared transmissions. In order to use this message with EPOSMote III, we need an external circuit for IR transmission. We used the Texas Instrument BOOST-IR Boosterpack 4, but the circuit is quite simple, it is shown below:



Texas Instrument BOOST-IR Boosterpack [4] IR Transmitter Circuit

For proper functioning we need to configure the carrier frequency. This setting varies between IR manufacturers: for Samsung the carrier frequency is 37.9 kHz and the duty cycle is 33.3%. Usually, a PWM generator is used to create this signal in microcontrollers. We made the same with EPOSMote III.
Thus, when we use the term logic 'level 1', it implies that the PWM is enabled, and when we use the term logic "level 0', it implies that the PWM is disabled (and the output pin signal is '0'). The 'Bit 0' denote an entire combination of 'level 1' and 'level 0' equal to the Samsung pattern that represents '0', the same for the 'Bit 1' and the Samsung pattern for '1'.
The oscilloscope pictures in the beggining of the page are the infrared sensor output, and they have the inverse logic level from the infrared emitter.
Thus we describe the sequence for a command message (Cool 21ºC Fan 2 Turbo Off Swing Off) as follow:

Sequence for a command message.

```
//Header
'level 1' for 800 us
'level 0' for 18000 us
'level 1' for 3000 us
'level 0' for 9000 us

//First data block
Byte(0x40)
Byte(0x49)
Byte(0xF0)
Byte(0x00)
Byte(0x00)
Byte(0x00)
Byte(0x0F)
'Bit 1'

//Interblock
'level 0' for 1460 us
'level 1' for 3040 us
'level 0' for 8900 us

//Second data block
Byte(0x80) //Byte 1
Byte(0x43) //Byte 2
Byte(0x7F) //Byte 3
Byte(0x8E) //Byte 4
Byte(0x06) //Byte 5
Byte(0x98) //Byte 6
Byte(0x0F) //Byte 7
'Bit 1'
```

We have three types of messages, the command message, the on message, and the off message. The command message is composed of two data blocks, the on and off messages have three data blocks. The on and off messages change according to the configuration parameters. We decided to use just one combination of parameters for the on and off messages, the A-C settings will be configured to Fan Mode, Fan Speed 1, Turbo Off, and Swing Off. The messages for on and off are shown below:

Sequence for a Turn On Message.

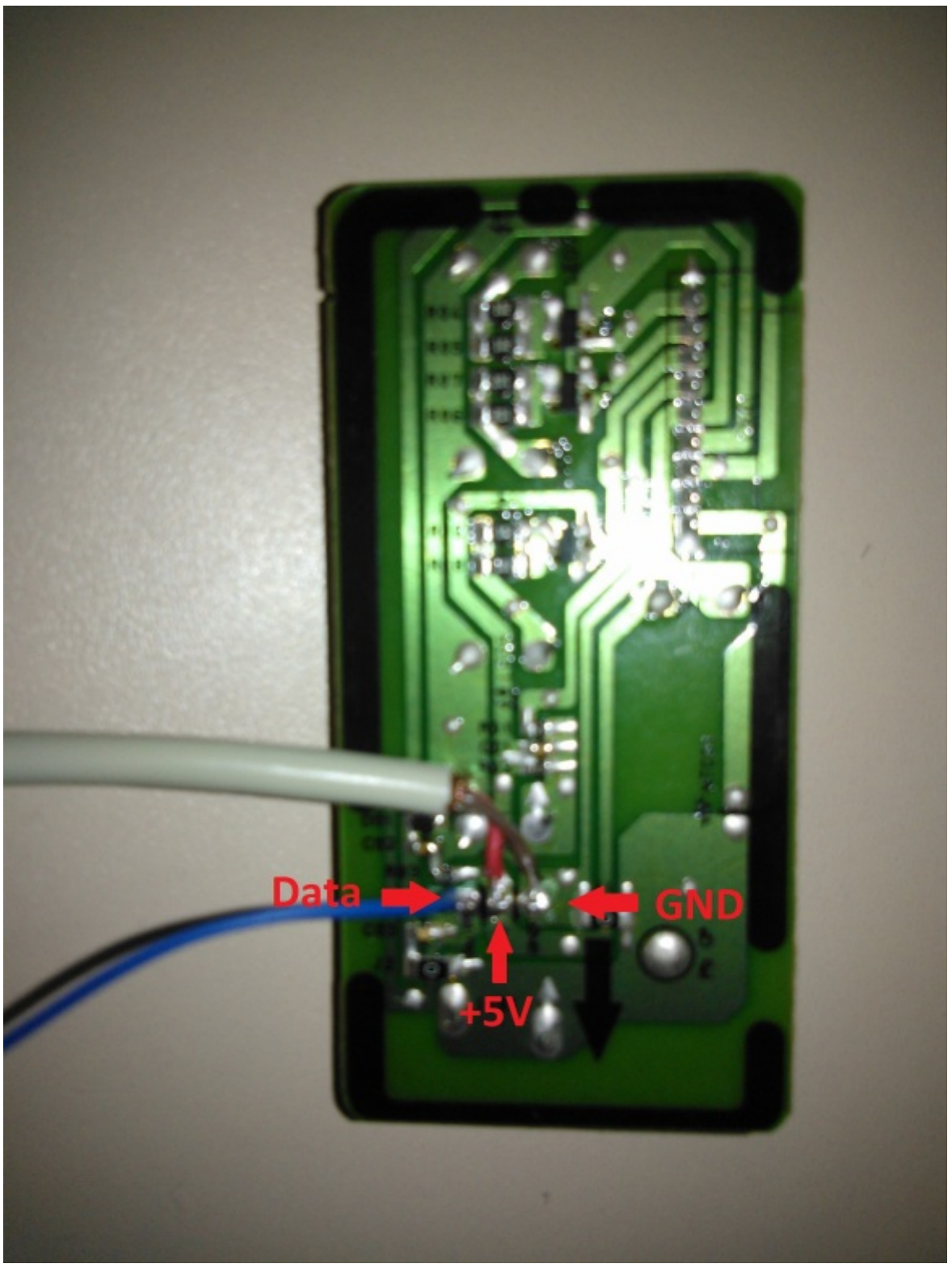
```
//Header
'level 1' for 800 us
'level 0' for 1800 us
'level 1' for 3000 us
'level 0' for 9000 us

//First data block
Byte(0x40)
Byte(0x49)
Byte(0xF0)
Byte(0x00)
Byte(0x00)
Byte(0x00)
Byte(0x0F)
'Bit 1'

//Interblock
'level 0' for 1460 us
'level 1' for 3040 us
'level 0' for 8900 us

//Second data block
Byte(0x80)
Byte(0x4B)
Byte(0xF0)
Byte(0x00)
Byte(0x00)
Byte(0x00)
Byte(0x00)
'Bit 1'

//Interblock
'level 0' for 1460 us
'level 1' for 3040 us
'level 0' for 8900 us

//Third data block
Byte(0x80)
Byte(0x43)
Byte(0x7F)
Byte(0x8E)
Byte(0x01)
Byte(0xAC)
Byte(0x0F)
'Bit 1'
```

Sequence for a Turn Off Message.

```
//Header
'level 1' for 800 us
'level 0' for 1800 us
'level 1' for 3000 us
'level 0' for 9000 us

//First data block
Byte(0x40)
Byte(0x4D)
Byte(0xF0)
Byte(0x00)
Byte(0x00)
Byte(0x00)
Byte(0x03)
'Bit 1'

//Interblock
'level 0' for 1460 us
'level 1' for 3040 us
'level 0' for 8900 us

//Second data block
Byte(0x80)
Byte(0x4B)
Byte(0xF0)
Byte(0x00)
Byte(0x00)
Byte(0x00)
Byte(0x00)
'Bit 1'

//Interblock
'level 0' for 1460 us
'level 1' for 3040 us
'level 0' for 8900 us

//Third data block
Byte(0x80)
Byte(0x47)
Byte(0x7F)
Byte(0x8E)
Byte(0x01)
Byte(0xAC)
Byte(0x03)
'Bit 1'
```

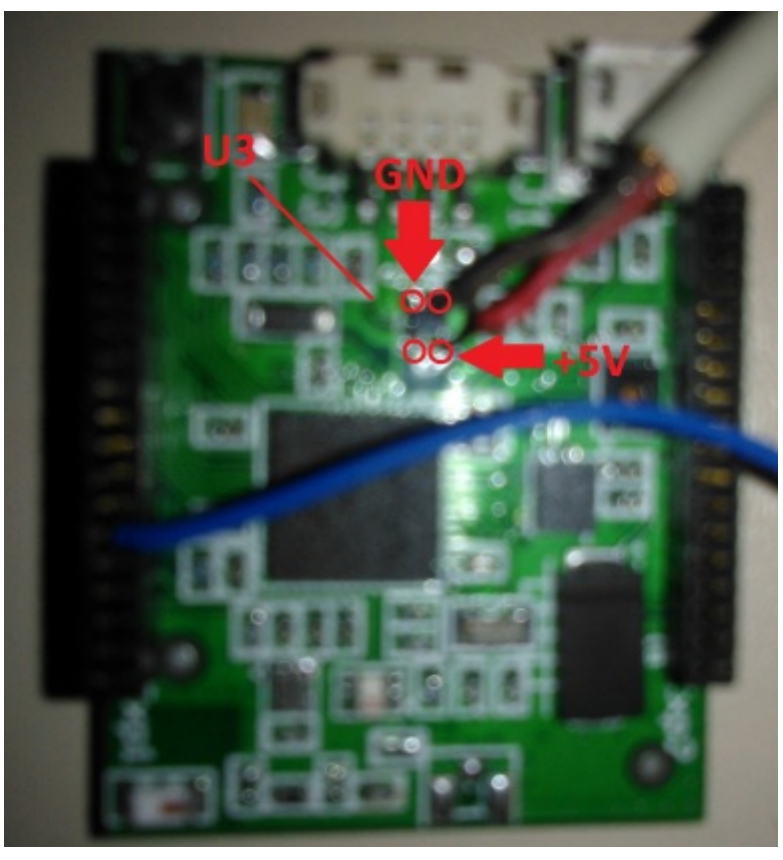
Using EPOSMote III to emulate the IR Controller inside the A-C

The other hardware connection option is to connect the EPOSMote III direct to the AC Receiver Board. We need to disconnect the infrared receiver and connect a wire direct to EPOSMote III. The figure below is a photograph of the AC Receiver Board with the EPOSMote III connections. In the other side of the PCB, we disconnected the IR Receiver Dout Pin. We used the pointed supply connection, because in these points the supply is always on, even in the AC stand by mode.



AC Receiver Board with EPOSMote III connections

In EPOSMote III board we need to connect the +5V before the LDO Regulator (U3). The figure below shows the connections for GND and +5V, the data wire is connected to PD2.



EPOSMote III connections

An advantage of this type of connection is that we do not need any external eletronic components to command the Samsung AC. Also, the signal after the infrared sensor is not modulated, i.e., we do not need to use PWM to generate the carrier signal.
The drawback is that the signal must be inverted, the logic 'level 0' is equal to +3V3 and the logic "level 1' is equal to 0V, as we can see in the oscilloscope figure at the beginning. This must be done in the 'Bit 1' and 'Bit 0', the code in the next section exemplify this changes.

EPOSMote III Code

--

The class Samsung_AC, depicted in the code below, is responsible for sending the commands from EPOSMote III to the Samsung Air Conditioners. The class supports both hardware methods to send commands: by PWM (IR controller) and directly connected to the AC board. The commented lines represent the commands sent when the IR controller is used. When EPOSMote III is directly connected to the AC board, an GPIO (D2) is used. The GPIO (or the PWM Timer) is created in the constructor.

Then, the enumerations define the Mode (COOL, DRY, FAN, HEAT, or AUTO), Fan mode (FAN1, FAN2, FAN3, FAN_AUTO1, or FAN_AUTO2), Turbo mode (ON or OFF), Swing mode (ON or OFF), and the Temperature (from 16 to 30). The enumeration values are defined according to the obtained values from the reverse engineering (see the previous Section).

The method **send_command** receives the mode, fan, temperature, turbo, and swing as parameters, mounts the message, calculates the CRC (the number of '1' in the message defines the CRC value), and send the bytes to the output (GPIO or PWM) by using the methods **send_byte** or **send_bit**. The method **send_byte** performs a loop in the received byte calling send_bit with the appropriate parameter ('1' or '0'). The method **send_bit** sends '1' or '0' to the output. It is important to highlight here that when EPOSMote III is directly connected to the AC board, the signal is inverted ('0' is high, and '1' is low). The methods **send_header** and **send_inter_block** are responsible for sending the header at the beginning of each message (it is fixed) and sending a block to indicate the end of block and the beginning of a new block (it is also fixed).

You can find two application examples in the LISHA svn: one that receives an AC command from the ZigBee and sends it to the AC 6 and another one that sends the ON, a command, and OFF periodically via ZigBee 7.

Class Samsung_AC responsible for sending commands to the Air Conditioner.

```
class Samsung_AC
{
public:
    Samsung_AC () {
        //_pwm = new eMote3_PWM(1, PWM_FREQUENCY, 0, 'd', 2);
        _d2 = new GPIO('d', 2, GPIO::OUTPUT);
    }

    enum Mode {
        COOL    = 0x08,
        DRY     = 0x04,
        FAN     = 0x0C,
        HEAT    = 0x02,
        AUTO    = 0x00,
    };

    enum Fan {
        FAN1            = 0x02,
        FAN2            = 0x01,
        FAN3            = 0x06,
        FAN_AUTO1       = 0x00, //Dry, Hot, Cool
        FAN_AUTO2       = 0x03, //AUTO Mode
    };

    enum Turbo {
        TURBO_ON      = 0x03,
        TURBO_OFF     = 0x00,
    };

    enum Swing {
        SWING_OFF     = 0x7F,
        SWING_ON      = 0xF5,
    };

    enum Temperature {
        T30 = 0x07,
        T29 = 0x0B,
        T28 = 0x03,
        T27 = 0x0D,
        T26 = 0x05,
        T25 = 0x09,
        T24 = 0x01,
        T23 = 0x0E,
        T22 = 0x06,
        T21 = 0x0A,
        T20 = 0x02,
        T19 = 0x0C,
        T18 = 0x04,
        T17 = 0x08,
        T16 = 0x00,
    };

    void send_command(Mode mode, Fan fan, Temperature temperature, Turbo turbo, Swing swing) {
        unsigned int tmp = mode << 17 | fan  << 14 | temperature <<  10 | turbo << 8 | swing;

        unsigned int count = 0;
        for(unsigned int i = 0; i < 21; i++) {
            if((tmp & (1 << i)))
                count++;
        }

        count += 5; //0x80 + 0x8E

        unsigned char crc;
        switch(count) {
            case 13:
                crc = 0x40;
                break;
            case 14:
                crc = 0x47;
                break;
            case 15:
                crc = 0x4B;
                break;
```

```
    case 16:
        crc = 0x43;

        break;
    case 17:
        crc = 0x4D;
        break;
    case 18:
        crc = 0x45;
        break;
    default:
        crc = 0x40;
        break;
}

//first block is always the same
send_header();
send_byte(0x40);
send_byte(0x49);
send_byte(0xF0);
send_byte(0x00);
send_byte(0x00);
send_byte(0x00);
send_byte(0x0F);
send_bit('1');

    send_inter_block();
    send_byte(0x80); //block header
    send_byte(crc); //CRC
    send_byte(swing); //Swing
    send_byte((turbo << 5) | 0x8E); //Turbo
    send_byte(temperature); //Temperature
    send_byte((fan << 4) | mode | 0x80); //Fan + Mode
    send_byte(0x0F);
    send_bit('1');
}

void turn_on() {
    send_header();
    send_byte(0x40);
    send_byte(0x49);
    send_byte(0xF0);
    send_byte(0x00);
    send_byte(0x00);
    send_byte(0x00);
    send_byte(0x0F);
    send_bit('1');

    send_inter_block();

    send_byte(0x80);
    send_byte(0x4B);
    send_byte(0xF0);
    send_byte(0x00);
    send_byte(0x00);
    send_byte(0x00);
    send_byte(0x00);
    send_bit('1');

    send_inter_block();

    send_byte(0x80);
    send_byte(0x43);
    send_byte(0x7F);
    send_byte(0x8E);
    send_byte(0x01);
    send_byte(0xAC);
    send_byte(0x0F);
    send_bit('1');
}

void turn_off() {
    send_header();
    send_byte(0x40);
    send_byte(0x4D);
    send_byte(0xF0);
    send_byte(0x00);
```

```
        send_byte(0x00);
        send_byte(0x00);

        send_byte(0x03);
        send_bit('1');

        send_inter_block();

        send_byte(0x80);
        send_byte(0x4B);
        send_byte(0xF0);
        send_byte(0x00);
        send_byte(0x00);
        send_byte(0x00);
        send_byte(0x00);
        send_bit('1');

        send_inter_block();

        send_byte(0x80);
        send_byte(0x47);
        send_byte(0x7F);
        send_byte(0x8E);
        send_byte(0x01);
        send_byte(0xAC);
        send_byte(0x03);
        send_bit('1');
    }

private:

    void send_byte(unsigned char byte) {
        unsigned int i;
        i = 8;
        while(i > 0) {
            if(byte & (1 << (i - 1)))
                send_bit('1');
            else
                send_bit('0');
            i--;
        }
    }

    //send bit 0 or 1 according to the parameter
    void send_bit(unsigned char bit) {
        //_pwm->set(PWM_FREQUENCY, 33);
        _d2->set(0);
        eMote3_GPTM::delay(560);
        //_pwm->set(PWM_FREQUENCY, 0);
        _d2->set(1);
        if(bit == '1')
            eMote3_GPTM::delay(1690);
        else
            eMote3_GPTM::delay(560);
    }

    void send_header() {
        //_pwm->set(PWM_FREQUENCY, 33);
        _d2->set(0);
        eMote3_GPTM::delay(800);
        //_pwm->set(PWM_FREQUENCY, 0);
        _d2->set(1);
        eMote3_GPTM::delay(18000);
        //_pwm->set(PWM_FREQUENCY, 33);
        _d2->set(0);
        eMote3_GPTM::delay(3000);
        //_pwm->set(PWM_FREQUENCY, 0);
        _d2->set(1);
        eMote3_GPTM::delay(9000);
    }

    void send_inter_block() {
        //_pwm->set(PWM_FREQUENCY, 0);
        _d2->set(1);
        eMote3_GPTM::delay(1460);
        //_pwm->set(PWM_FREQUENCY, 33);
        _d2->set(0);
```

```
        eMote3_GPTM::delay(3040);
        //_pwm->set(PWM_FREQUENCY, 0);

        _d2->set(1);
        eMote3_GPTM::delay(8900);
    }

private:
    //eMote3_PWM * _pwm;
    GPIO * _d2;
};
```

--

1 IRremote Arduino Library - https://github.com/z3t0/Arduino-IRremote

2 Samsung Application Note - IR Remote Controller - http://elektrolab.wz.cz/katalog/samsung_protocol.pdf

3 AnalysIR Blog - http://www.analysir.com/blog/

4 Boost-IR Boosterpack EStore TI - https://store.ti.com/boost-ir.aspx

5 Tiva Laucnhpad - https://store.ti.com/Tiva-C-LaunchPad.aspx

6 AC receiver application - https://svn.lisha.ufsc.br/openepos/epos2/branches/smart_building/app/samsung_ac_receiver.cc

7 AC sender application - https://svn.lisha.ufsc.br/openepos/epos2/branches/smart_building/app/samsung_ac_sender.cc