DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

RESEARCH AND IMPLEMENTATION OF UNIFIED SMART ADAPTIVE REMOTE CONTROL PROTOCOL FOR CONSUMER ELECTRONIC EQUIPMENTS

Ahmet Selçuk ÖZTÜRK

December, 2006
İZMİR

RESEARCH AND IMPLEMENTATION OF UNIFIED SMART ADAPTIVE REMOTE CONTROL PROTOCOL FOR CONSUMER ELECTRONIC EQUIPMENTS

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

by
Ahmet Selçuk ÖZTÜRK

December, 2006
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"RESEARCH AND IMPLEMENTATION OF UNIFIED SMART ADAPTIVE REMOTE CONTROL PROTOCOL FOR CONSUMER ELECTRONIC EQUIPMENTS "** completed by **Ahmet Selçuk ÖZTÜRK** under supervision of **Prof. Dr. Alp KUT** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Alp KUT

Supervisor

Doç.Dr.Yalçın ÇEBİ

(Jury Member)

Yard.Doç.Dr.Zafer DİCLE

(Jury Member)

Prof.Dr. Cahit HELVACI

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like to thank to my supervisor, Prof. Dr. Alp KUT for his guidance and assistance in this thesis.

I would like to thank to Assoc. Prof. Dr. Yalçın ÇEBİ for his guidance to improve this thesis content and its structure.

I would like to thank to Asst. Prof. Dr. Zafer DİCLE who has listened, encouraged and motivated me when I was exhausted during end of this thesis.

I also thank to my colleagues and managers from BEKO Electronic R&D departments for their endless support and encouragements.

Finally, I would like to thank to my family for their endless support and encouragements.

Ahmet Selçuk ÖZTÜRK

RESEARCH AND IMPLEMENTATION OF UNIFIED SMART ADAPTIVE REMOTE CONTROL PROTOCOL FOR CONSUMER ELECTRONIC EQUIPMENTS

ABSTRACT

Continuously growing technology and changes in customer needs has brought together a vast increase in the number of consumer electronic devices. With each new device, new features are introduced in parallel to this growing technology. New features mean more buttons on the devices' remote controllers.

Remote controllers were unable to show the same technological improvements as the new technology consumer electronic devices. Remote controller concepts in use today enable us to design only controllers with fixed number of buttons. Nevertheless some remote controllers that exist in the market today can define their own buttons and even macros by their properties or by aid of the computers. But all remote controllers basically control the devices with single directional communication by sending key codes with the help of protocols they use.

To realize the same technological development achieved in consumer electronic devices a new concept is needed for their remote controllers. This approach is based on the principle of changing the number of keys on the remote controller that can be used according to the state of the controlled device.

The aim of this study is to design a new protocol in order to develop more flexible applications and provide a framework for new kind of applications. In this approach bidirectional communication is targeted. The main aim in this thesis is making the communication is bidirectional. By doing this the number of key can be reduced or limited. Thus users will deal with smaller number of keys consequently the system will be less likely to crash on any given condition as the user will not be able to press unsuitable keys.

Keywords: Remote Controllers, Remote Control Protocols, Adaptive Remote Controllers

TÜKETİCİ ELEKTRONİĞİNE YÖNELİK TEKİL, AKILLI VE ADAPTE OLUP FARKLILAŞABİLEN UZAKTAN KUMANDA PROTOKOLÜNÜN ARAŞTIRILMASI VE TASARIMI

ÖZ

Hızla artan teknoloji ve ihtiyaçların değişmesi ile beraber günlük yaşantımıza giren tüketici elektroniği cihazlarının sayısı artmaktadır. Her yeni cihaz, gelişen teknolojiye paralel olarak yeni özellikleride beraberinde getirmektedir. Daha fazla özellik bu cihazları kontrol etmek için tasarlanmış kumandalarda da daha fazla tuş anlamına gelmektedir.

Yeni teknoloji tüketici elektroniği cihazlarının gösterdiği teknolojik ilerlemeyi kumandalar gösterememiştir. Günümüzde kullanılan uzaktan kumanda yapısı ile üzerlerinde belirli sayıda tuşlardan oluşan kumandalar tasarlanmaktadır. Yine bilgisayar ya da kendi özellikleri ile tuş ve makro tanımlanabilen bir takım uzaktan kumandalar da mevcuttur. Fakat hepsi, tek yönlü bir iletişim ile, tuş kodları kullanarak, kontrol edilen cihazları, kullandıkları protokol yardımiyle kontrol etmektedir.

Tüketici elektroniği cihazlarındaki gelişmeyi, onları kontol eden uzaktan kumandalarda da gerçekleştirebilmek için yeni bir yaklaşım gerekmektedir. Bu yaklaşım, kumanda üzerindeki kullanılabilir tuş sayısının, kontrol edilen cihazın bulunduğu durumuna gore değiştirilmesi esasına dayanmaktadır.

Bu çalışmanın amacı, daha esnek ve yeni uygulamaların geliştirilmesine imkan sağlayan bir iskelet oluşturacak, yeni yaklaşımı uygulanabilir duruma getirecek protokol tasarlamaktır. Bu yaklaşımda iletişimin çift yönlü olması amaçlanmıştır. Bunun amacı belirli bir zamanda kumanda üzerindeki tuş sayısını azaltmak ya da kullanılabilecek tuş sayısını sınırlandırmaktır. Böylelikle kullanıcı hem az tuş ile karşılaşacak hem de kontrol edilen cihazın bulunduğu durumda kullanılmayacak bir tuşa basması ile cihazı kararsız hale getirmesi önlenecektir.

Anahtar Kelimeler : Uzaktan Kumandalar, Uzaktan Kumanda Protokolleri, Adapte Olabilen Kumandalar.

CONTENTS **Page**

CHAPTER ONE

INTRODUCTION

1.1 Introduction

Since the time remote controllers were invented and used in consumer products they started to be very important part of these products over the course of time. Remote controllers when first designed were targeted only at controlling consumer products. And products were not complex also were not so many. People had at most a TV set and may be a VCR. For this reason remote controllers of those days were only designed to control one device and were not that complex. But now the situation is dramatically changed.

There has been a great increase at the variety and number of products owned by people. Nowadays a TV set, a music set a VCR or DVD Player/Recorder and may be some other different consumer products can be seen in any house. And all these products have their own remote controllers which have many complex functions to perform. For this reason remote controllers' concept has changed. But the controller protocols and medium have not changed especially for consumer electronic products.

A different approach for remote controllers that supports new requirements is now needed. These requirements are being adaptive, error tolerant, generally buttonless and platform independent. These requirements are explained below.

1.2 Requirements

The main requirements are to make remote controllers very flexible/adoptable, easy to use and these which include mechanisms that would prevent errors both for the controller and the controllable device and which would decrease the number of remote controller units to one.

1.2.1 Adaptive Remote Controller

Remote controllers contain certain number of keys to control any specific device. If the device has a complex structure the numbers of remote controllers' keys are increased. This affects the usability of that controller. For that reason remote controllers' keys should be easily increased or decreased but they should only contain required keys at any time depending on the controllable device state.

1.2.2 Error Tolerance

Once the remote controller contains the keys for only one state at a time, systems will not be able to accept any key that makes it crash and they will not crash the system during state changes by sending unusable keys from previous state.

1.2.3 Buttonless Design

To be adaptive, remote controllers must be able to change its button structures easily and present them to the user by different layouts. This is the property of presentation layer which may be able to learn user's addictions and remote controller may be able to serve the buttons to touch screen depending on those addictions by drawing some of them bigger or smaller than others or in different color by using a touch screen.

1.2.4 Platform Independency

Remote controllers enable the users to control all devices that are compatible with this concept. In order to provide this, there must be a common protocol. Thus users will not deal with many remote controllers and will not keep many remote controllers in hand to control any device when needed. In addition to this, consumer electronic producers can design generic remote controller for their produced any device by supporting specific protocol.

1.3 Structure of the Thesis

The contents of this thesis are constructed according to this outline.

In Chapter 2, the communication mediums are described which are used today.

In Chapter 3, current remote controller protocols are examined.

Chapter 4 is an introduction to Smart Adaptive Remote Controller concept, synchronization mechanism and its requirements.

Chapter 5 contains definitions of system and messaging states.

Chapter 6 contains definitions of new protocol structure and communication packets.

Chapter 7 contains definitions of entities and their types which are used in Smart Adaptive Remote Controller Protocol.

Chapter 8 gives brief information about simulation application of Smart Adaptive Remote Controller Protocol.

Chapter 9 presents conclusion and future work.

CHAPTER TWO

EXISTING REMOTE CONTROLLER COMMUNICATION

2.1 Existing Remote Controllers Approach

Nowadays remote controllers are classified by physical communication mediums, their protocols and types. Type differentiates the remote controllers by the number of devices to be controlled. These types of remote controllers which control many devices are called universal remote controllers. These remote controllers are bought from electronic stores. Others are provided together with new products and they control only those products.

The other classification element is communication medium and the most used communication medium in consumer electronic is infrared technology and this is what is explained in this chapter.

Today, the common properties of existing remote controllers are working only in one direction, the signals or communications transfer the information only in one direction from remote controller to controllable device one at a time. One of the reasons of this is using infrared technology as a communication medium and infrared technology is mostly used in this communication.

Another communication medium is Radio Frequency which is rarely used for specific applications. The following title gives a brief information about communication mediums.

2.2 Communication Mediums

The communication mediums mostly used today are infrared technology for remote controller devices and rarely radio frequency can also be used. If the medium

is radio frequency the method or its applications can be Bluetooth or zigBee which is newly developed. But in consumer electronic sector remote controllers' medium is % 99.99 infrared (IR) technologies. For this reason only IRDA physical part is described here.

2.3 Infrared

Infrared is a legacy technology that will not die any time soon. Infrared is a wireless communication technology that makes use of the invisible spectrum of light that is just beyond red in the visible spectrum. It's suitable for applications that require short-range, point-to-point data transfer. Because it uses light, line of sight is a prerequisite for using infrared. Despite this limitation, infrared is widely used in household equipment and is increasingly popular in devices such as digital cameras, PDAs, and notebook computers. (Dumbill, Jepsoon & Weeks, 2004)

Founded in 1993 as a nonprofit organization, the Infrared Data Association (IrDA) is an international organization that creates and promotes interoperable, low-cost infrared data interconnection standards that allow users to transfer data from one device to another. The Infrared Data Association standards support a broad range of appliances, computing, and communications devices.

There are currently four versions of IrDA; their differences are mainly in the transfer speed:

- **Serial Infrared (SIR)**
  The original standard with a transfer speed of up to 115 kbps
- **Medium Infrared (MIR)**
  Improved transfer speed of 1.152 Mbps; it is not widely implemented
- **Fast Infrared (FIR)**
  Speed of up to 4 Mbps; most new computers implement this standard
- **Very Fast Infrared (VFIR)**
  Speed of up to 16 Mbps; it is not widely implemented yet

When two devices with two different IrDA implementations communicate, one steps down to the lower transfer speed.

2.3.1 IRDA Physical Part

The IrDA Physical Layer Specification sets a standard for the IR transceiver, the modulation or encoding/decoding method and also other physical parameters. IrDA uses IR with peak wavelength of 0.85 to 0.90μm. The transmitter's minimum and maximum intensity is 40 and 500 mW/Sr within a 30 degree cone. The receiver's minimum and maximum sensitivity is 0.0040 and 500 mW/cm^2 within a similar 30 degree cone. The link length is 0 to 1m with an error rate of less than 1 in 10^8 bits.

There are three different modulations or encoding/decoding methods. The first one is mandatory for both IrDA-1.0 and IrDA- 1.1. The other two are optional and are for IrDA-1.1 only. For transfer rate of 9.6k, 19.2k, 38.4k, 57.6k or 115.2kbps operations, a start (0) bit and a stop (1) bit is added before and after each byte of data. This is the same format as used in a traditional UART. However, instead of NRZ (non return to zero), a method similar to RZ (return to zero) is used, where a 0 is encoded as a single pulse of 1.6μsec to 3/16 of a bit cell, and a 1 is encoded as the absence of such a pulse. In order to have unique byte patterns to mark beginning and ending of a frame and yet allow any binary data bytes, byte stuffing (escape sequence) is used in the body of the frame.

A 16-bit CRC is used for error detection. The 9.6kbps operation is mandatory for both IrDA-1.0 and IrDA-1.1. 19.2k, 38.4k, 57.6k and 115.2 kbps are all optional for IrDA-1.0 and IrDA-1.1. For transfer rate of 0.576M or 1.152 Mbps operation, no start or stop bits are used and the same synchronous format as HDLC is used. Again, a 0 is encoded as a single pulse (1/4 the bit cell) whereas a 1 is encoded as the absence of such a pulse. In order to ensure clock recovery, bit stuffing is used (same as in HDLC). The same 16-bit CRC is also used. Both 0.576M and 1.152 Mbps operations are optional for IrDA-1.1. For transfer rate of 4.0 Mbps operation, a 4-PPM method is used. Again, no start or stop bits are used. In addition, bit/byte

stuffing is not needed either. A 32- bit CRC is used in this case. This rate is used in IrDA-1.1 only. (Yeh, K.W., Wang, L.).

CHAPTER THREE

IR PROTOCOLS

3.1 Introduction

There are several remote controller protocols currently used. The manufacturers which produce consumer products use their own IR code formats or protocols. The reason of this is by using their own genuine remote controller system to guarantee their products controller signals never interfere with other products which are produced by other consumer electronic manufacturers. This is a prestige for manufacturers. For that reason there are many different remote controller protocols in this area. The well known protocols are Philips's RC5, Sony's, NEC's and Matsushita's IR code formats. But most popular one is Philips's RC5 protocol. Except RC5 protocol, other protocols are not called protocol so much, they are called code format.

3.2 Philips's RC5 IR Protocol

The RC5 remote controller protocol is one of the most popular and is widely used to control numerous home appliances, entertainment systems and some industrial applications including utility consumption remote meter reading, contact-less apparatus control, telemetry data transmission, and car security systems. Philips originally invented this protocol and virtually all Philips' remotes use this protocol.

Following is a description of the RC5. When the user pushes a button on the hand-held remote, the device is activated and sends modulated infrared light to transmit the command. The remote separates command data into packets. Each data packet consists of a 14-bit data word, which is repeated if the user continues to push the remote button. The data packet structure is as follows:

2 start bits, 1 control bit, 5 address bits, 6 command bits.

The start bits are always logic ‘1’ and intended to calibrate the optical receiver automatic gain control loop. Next, is the control bit. This bit is inverted each time the user releases the remote button and is intended to differentiate situations when the user continues to hold the same button or presses it again. The next 5 bits are the address bits and select the destination device. A number of devices cause RC5 at the same time. To exclude possible interference, each must use a different address. The 6 command bits describe the actual command. As a result, a RC5 transmitter can send the 2048 unique commands.

The transmitter shifts the data word, applies Manchester encoding and passes the created one-bit sequence to a control carrier frequency signal amplitude modulator.(Seerden, P.) The amplitude modulated carrier signal is sent to the optical transmitter, which radiates the infrared light. In RC5 systems the carrier frequency has been set to 36 kHz. Figure 3.1 displays the RC5 protocol. The receiver performs the reverse function. The photo detector converts optical transmission into electric signals, filters it and executes amplitude demodulation. The receiver output bit stream can be used to decode the RC5 data word. This operation is done by the microprocessor typically. Single-die optical receivers are being mass produced by a number of companies such as Siemens, Temic, Sharp, Xiamen Hualian, Japanese Electric and others. (Kremin, n.d.)

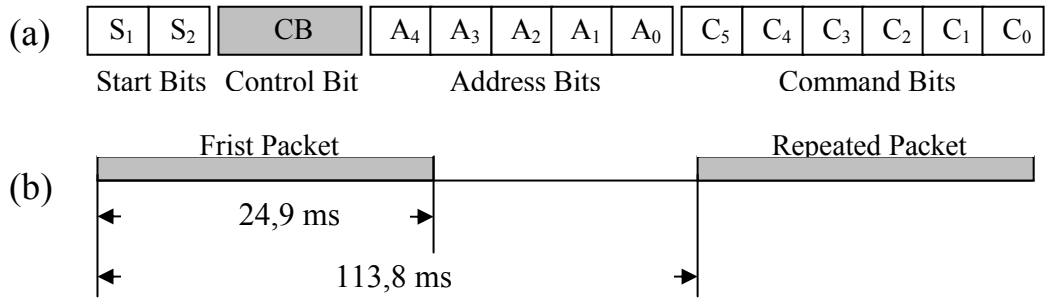


Figure 3.1 (a) RC5 protocol encoded data word (b) Consecutive data packets

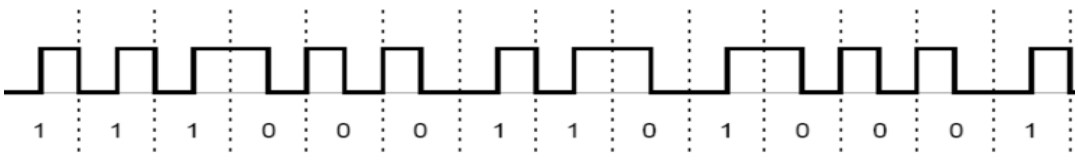


Figure 3.2 RC5 protocol encoded data word

3.3 SONY IR Protocol

The Sony IR protocol has three different versions: 12-bit, 15-bit and 20-bit versions. It is assumed that the 15-bit and 20-bit versions differ in the number of transmitted bits per command sequence in theory. But there are not found detailed information about these two types. For that reason described here 12-bit version Sony IR protocol.

The Sony remote controller is based on the Pulse-Width signal coding scheme. The code exists of 12 bits sent on a 38kHz carrier wave. The data packet structure is as follows:

1 start bit, 7 command bits, 5 address bits or device code

The code starts with a header of 2,4ms or 8T where T is 300µs. The header is followed by 7 command bits that indicating an action to be performed and these command code bits are followed by 5 address bits that indicating which device should act upon the command code. When data are transmitted repeatedly, the frame cycle is 45ms or 150 period. This means that total length of a bit stream is always 45ms.

The address and commands exists of logical ones and zeros. A logical one is formed by a space of 600µs or 2T and a pulse of 1200 µs or 4T. A logical zero is formed by a space of 600 µs or 2T and pulse of 600µs or 2T. The space between 2 transmitted codes when a button is being pressed is 40ms. The bits are transmitted least significant bits first. (Celadon, n.d.)

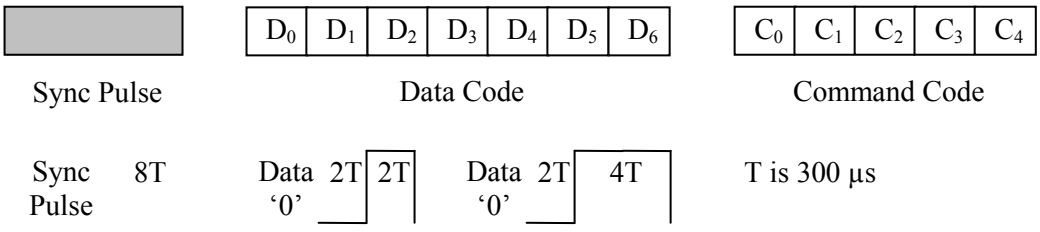


Figure 3.3 Sony Protocol

The encoded data, 600µs for 0's(zero's), 1.2ms for 1's or 2.4ms for start bit on periods followed by 600µs off periods or space, is then used to modulate a 38kHz signal which is used to drive the IR LED.

When the encoded signal is on, the 38kHz signal is transmitted, when the encoded signal is off, the 38kHz signal is not transmitted, this gives a 'modulated' 38kHz signal which is actually used to drive the IR LED. (SB-Project, n.d.)

3.4 Philips RECS-80 IR Protocol

This protocol is designed by Philips and transmitters are produced by Philips and ST Microelectronics. Actually it was not seen this protocol being used in real applications up to now that our professional work experiment. For that reason information on this page is derived from the some data sheets of the Philips, ST microelectronics and researches on Internet.

There are two small differences between the two company's IC. The Philips IC has two modes of operation, one which is compatible with the ST chip and other one can handle up to 20 sub-system addresses. (SAA3008, 1988).

The protocol features are;

- 7 or 20 sub-system addresses, 64 commands per sub-system address
- 1 or 2 toggle bits to avoid key bounce
- Pulse distance modulation
- Carrier frequency of 38kHz, or unmodulated
- Bit time logic "0" is 5.1ms, logic "1" is 7.6ms (@ 455kHz Oscillator)
- Command repetition rate 121.5ms (55296 periods of the main oscillator)

The protocol uses pulse distance modulation. A logic "0" consists of a mark with a length of 6 periods of the carrier frequency (158µs), followed by a space which makes the total pulse to pulse distance 5.06ms. A logic "1" consists of a mark with a length of 6 periods of the carrier frequency, followed by a space which makes the total pulse to pulse distance 7.59ms. The Figure 3.4 represents these situations.

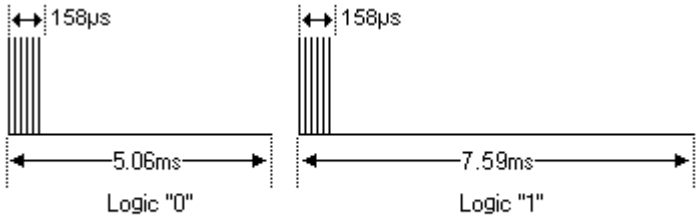


Figure 3.4 RECS-80 bit abbreviations

The Figure 3.5 shows a typical pulse train of a normal RECS-80 message. This example transmits command 36 to address 4.

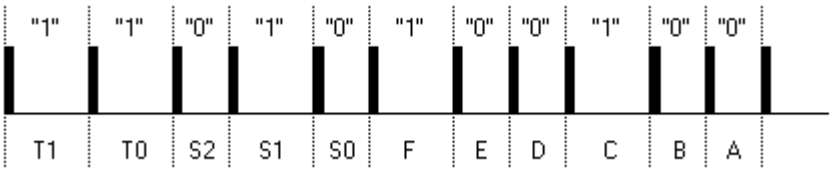


Figure 3.5 RECS-80 pulse train

Usually the first pulse is a reference pulse, with a value of "1". The receiver may use this bit to determine the exact bit length. The next bit is a toggle bit. Its value is toggled whenever a key is released, which results in a different code every time a new key is pressed. This allows the receiver to discriminate between new key presses and key repetitions. Only the ST chip M3004 can disable its carrier, in which case the REF pulse is interpreted as a second toggle bit.(STMA3004LD, 2004) The 2-bit toggle value is incremented every time a key is released. Thus only in this mode there is no real REF pulse. The next 3 pulses S2 to S0 represent the sub-system address bits, sent with MSB first. This would allow for 8 different sub-system addresses but both the SAA3008 and the M3004 can only generate 7 sub-system addresses in normal mode. Next come the 6 command bits F to A, also sent with MSB first allowing for 64 different commands per sub-system address. The pulse train is terminated by a last pulse; otherwise there is no way to know the duration of bit A. (SAA3008, 1988)

The entire command is repeated (with unchanged toggle bits) for as long as the key is held down. The repetition rate is 121.5ms (55296 periods of the oscillator).

Address assignments are a bit odd with this protocol. You can not simply convert the binary value to a decimal value. Below you see a table explaining the relationship between the binary and decimal sub-system address values.

Table 3.1 RECS-80 Conversion Table for 7 sub-system

Binary	Decimal
1 1 1	1
0 0 0	2
0 0 1	3
0 1 0	4
0 1 1	5
1 0 0	6
1 0 1	7

If more than 7 sub-system addresses needed application engineers can use the extended protocol which allows 13 additional sub-systems addresses. This is possible only if used SAA3008. The Figure 3.6 shows an extended message. This example transmits command 36 to address 10.

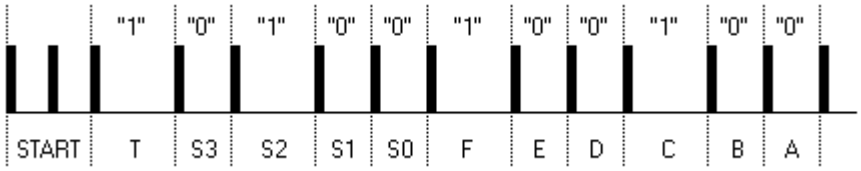


Figure 3.6 RECS-80 pulse train for extended protocol

The first two pulses are a special start sequence. The total duration of these pulses is equal to a normal "1" period. The next bit is a toggle bit. Its value is toggled whenever a key is released, which results in a different code every time a new key is pressed. This allows the receiver to discriminate between new key presses and key repetitions. The next 4 pulses S3 to S0 represent the sub-system address bits. This would allow for an additional 16 different sub-system addresses, although the

SAA3008 can only generate 13 additional sub-system addresses in this mode. Next come the 6 command bits F to A, also sent with MSB first. The pulse train is terminated by a last pulse; otherwise there is no way to know the duration of bit A. The entire command is repeated (with unchanged toggle bits) for as long as the key is held down. The repetition rate is 121.5ms (55296 periods of the oscillator).

Address assignments are a bit odd with this protocol. You can not simply convert the binary value to a decimal value. Below you see a table explaining the relationship between the binary and decimal sub-system address values. (SB-Project, n.d.)

Table 3.2 RECS-80 Conversion Table for 12 sub-system

Binary	Decimal
0 0 0 0	8
1 0 0 0	9
0 1 0 0	10
1 1 0 0	11
0 0 0 1	12
1 0 0 1	13
0 1 0 1	14
1 1 0 1	15
1 0 1 0	16
0 1 1 0	17
1 1 1 0	18
0 1 1 1	19
1 1 1 1	20

3.5 NEC IR Protocol

This protocol was developed by NEC. I've seen very similar protocol descriptions on the internet, and there the protocol is called Japanese Format. NEC manufactured the remote controller IC. But many IC manufacturers which produce controller can be produce their products to be decoded NEC format if ordered high volume by special agreements.

The protocol feature;

- 8 bit address and 8 bit command length
- Address and command are transmitted twice for reliability
- Pulse distance modulation
- Carrier frequency of 38kHz
- Bit time of 1.12ms or 2.25ms

The NEC protocol uses pulse distance encoding of the bits. Each pulse is a 560µs long 38kHz carrier burst (about 21 cycles). A logical "1" takes 2.25ms to transmit, while a logical "0" is only 1.12ms. The recommended carrier duty-cycle is 1/4 or 1/3.

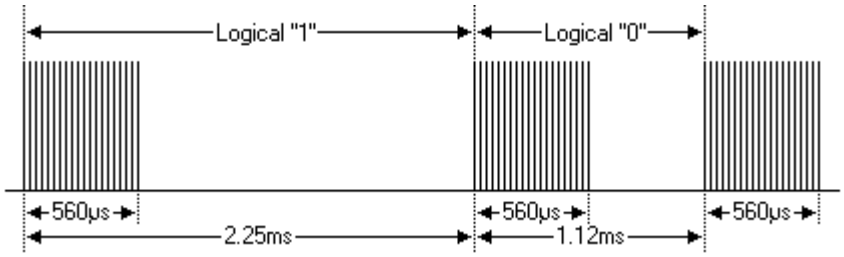


Figure 3.7 NEC Modulation

The Figure 3.8 shows a typical pulse train of the NEC protocol. With this protocol the LSB is transmitted first. In this case Address $59 and Command $16 is transmitted. A message is started by a 9ms AGC burst, which was used to set the gain of the earlier IR receivers. This AGC burst is then followed by a 4.5ms space, which is then followed by the Address and Command. Address and Command are transmitted twice. The second time all bits are inverted and can be used for verification of the received message. The total transmission time is constant because every bit is repeated with its inverted length. If you're not interested in this reliability you can ignore the inverted values, or you can expand the Address and Command to 16 bits each!

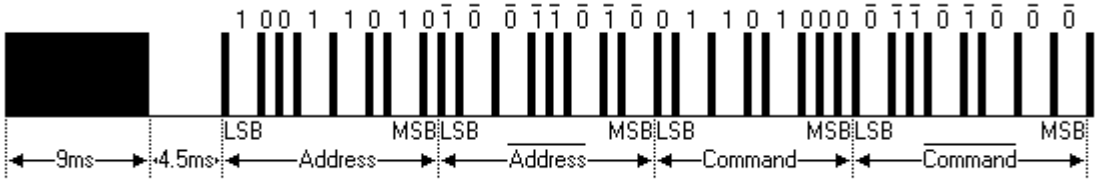


Figure 3.8 NEC Protocol Pulse Train

A command is transmitted only once, even when the key on the remote controller remains pressed. Every 110ms a repeat code is transmitted for as long as the key remains down. This repeat code is simply a 9ms AGC pulse followed by a 2.25ms space and a 560µs burst. (SB-Project, n.d.)

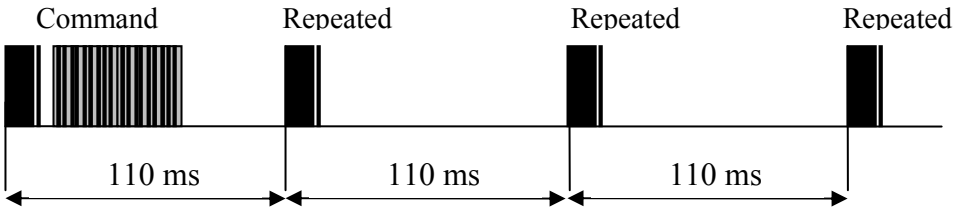


Figure 3.9 Repeated commands wave form

3.6 Panasonic's IR Protocol

Another infrared remote protocol I will explain is an older Panasonic remote protocol. The protocol is similar with the RECS-80 protocol but it uses more bits than the RECS-80 protocol. For the data transmission Panasonic uses the pulse-place modulation.

For the communication a pulse is used with a fixed length, followed by which represents the logical state of the bit. 2048 codes are defined in this protocol, divided in 5 bits of custom code and 6 bits of data code. The custom code is a value which represents the manufacturer code and the data code is a value which represents the pressed button on the remote controller.

The full transmitted code is 22 bits: First a header is sent then the custom code (5 bits), then the data code, followed by the inverse of the custom code and the inverse of the data code, and to terminate a stop bit is added to the code. The inverse transmitted bits are very useful for the error detection.

Each first part of a bit is always a high level with a fixed time and is followed by a low level where the time defines if the bit is a logic 1 or a logic 0.

Timing diagram:

T =420 µs to approx 424 µs in the USA and Canada

T=454 µs to approx 460 µs in Europe and others

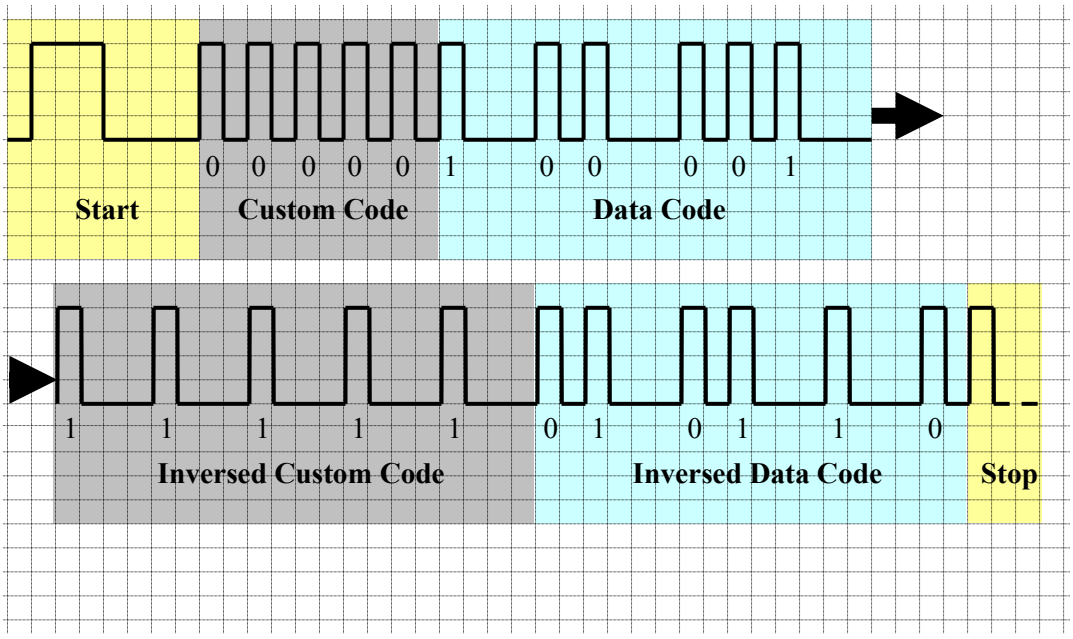


Figure 3.10 The Panasonic IR Protocol Pulse Train

The header is 8T high and 8T low, a 1 is coded 2T high and 6T low, a 0 is coded 2T high and 2T low.

3.7 RCMM (Remote Control Multimedia) IR Protocol

This protocol developing by Philips for interactive multimedia remote controller products like wireless keyboards, mice and game pads for that reason the commands had to be short and very low power consumption to maximize remote usage and main product performance. (SB-Project, n.d.)

Although RC5 and RC6 is dedicated to remote controller type of applications, it is not ideally suited for more interactive new input devices like wireless keyboards, multiple wireless game devices and wireless pointing devices.

Therefore Philips has defined a higher speed and very low power consumption protocol targeted to the above new products.

Although it is a standard within Philips, it is currently widely used as a 'de facto standard' for many OEM customers. RCMM can either work in a one-way or a two-way mode depending on whether multiple simultaneous devices need to work together or not.

Philips patented and published RCMM 1.5 version. This version is designed to provide the decoding simplicity of a one-way protocol and at the same time allow for multi-user simultaneous use, e.g. gamepads. In short it combines the best out of both worlds. No additional complexity is introduced at the receiver side, while it opens up a plethora of new possibilities in the multi-user application areas. Typical applications include set-top-box remote controllers or gamepads that allow multi-player gaming.

The RCMM protocol feature;

- 12 bits or 24 bits per message
- Pulse position coding, sending 2 bits per IR pulse
- Carrier frequency of 36kHz
- Message time ranges from 3.5 to 6.5 ms, depending on data content
- Repetition time 28 ms (36 messages per second)

The Figure 3.11 shows the most important transmission times of RCMM protocol. The message time is the total time of a message, counting form the beginning of the first pulse until the end of the last pulse of the message. This time can be 3.5 to 6.5 ms, depending on the data content and protocol used.

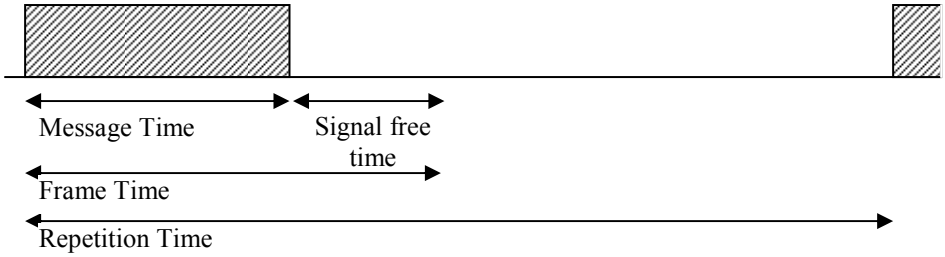


Figure 3.11 The RCMM Transmission time

The signal free time is the time in which no signal may be sent to avoid confusion with foreign protocols on the receiver's side. Philips recommends 1 ms for normal use, or 3.36 ms when used together with RC-5 and RC-6 signals.

The frame time is the sum of the message time and the signal free time, which can add up to just about 10 ms per message. About the repetition time is the recommended repetition time of 27.778 ms, which allows 36 messages per second. This is only a recommendation and is mainly introduced to allow other devices to send their commands during the dead times.

No provision is made for data collisions between two or more remote controllers! This means that there is no guarantee that the messages get across.

With this protocol a 36 kHz carrier frequency is used to transmit the pulses. This helps to increase the noise immunity at the receiver side and at the same time it reduces power dissipated by the transmitter LED. The duty cycle of the pulses is 1/3 or 1/4.

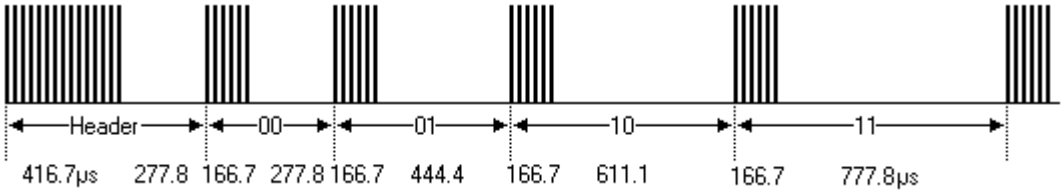


Figure 3.12 The RCMM pulse train

Each message is preceded by a header pulse with the duration of 416.7 µs (15 pulses of the carrier), followed by a space of 277.8 µs (10 periods of the carrier). This header is followed by 12 or 24 bits of data.

By changing the distance between the pulses two bits of data are encoded per pulse. Below you find a table with the encoding times.

Table 3.3 RCMM encoding times

Data	Mark	Space
0 0	166.7 µs (6 cycles)	277.8 µs (10 cycles)
0 1	166.7 µs (6 cycles)	444.4 µs (16 cycles)
1 0	166.7 µs (6 cycles)	611.1 µs (22 cycles)
1 1	166.7 µs (6 cycles)	777.8 µs (28 cycles)

The RCMM protocol has three different type that called modes. Each mode is intended for a particular purpose and differs mainly in the number of bits which can be used by the application. All data is sent with MSB first.

3.7.1 RCMM 12 Bit Mode

The 12 bit mode is the basic mode, and allows for 2 address bits and 8 data bits per device family. There are 3 different device families defined: keyboard, mouse and game pad. The Figure 3.13 describes 12 Bit mode packet structure.

Mode	Adress	Data
2 bits	2 bits	8 bits

Figure 3.13 12 Bit mode packet structure

The 12 bit mode is the basic mode, and allows for 2 address bits and 8 data bits per device family. There are 3 different device families defined: keyboard, mouse and game pad.

Table 3.4 Mode bits for RCMM 12 Bit Mode

Mode bits	Device Type
0 0	Extended mode
0 1	Mouse mode
1 0	Keyboard mode
1 1	Game pad mode

The 2 address bits provide for a way to use more than 1 device simultaneously. The data bits are the actual payload data.

3.7.2 RCMM 24 Bit Mode

The 24 bit mode, also known as extended mode, and allows more data to be transmitted per message. For instance for multi-lingual keyboards or a high resolution mouse.

Figure 3.14 24 Bit mode packet structure

Table 3.5 Mode bits for RCMM 24 Bit Mode

Mode bits	Device Type
0 0 0 0	OEM mode
0 0 0 1	Extended Mouse mode
0 0 1 0	Extended Keyboard mode
0 0 1 1	Extended Game pad mode

3.7.3 RCMM OEM Mode

In the OEM mode the first 6 bits are always 0 0 0 0 1 1. The next 6 bits are the customer ID (OEM manufacturer). My observation showed that Nokia used the code 1 0 0 0 0 0 for their 9800 series digital satellite receivers. Finally the last 12 bits are the actual pay load data.

Mode	Customer ID	Data
6 bits	6 bits	12 bits

Figure 3.15 24 Bit mode packet structure

3.8 Philips's RC6 IR Protocol

RC-6 is, as may be expected, the successor of the RC-5 protocol. Like RC-5 the new RC-6 protocol was also defined by Philips. It is a very versatile and well defined protocol. Because of this versatility its original definition is many pages long. Here on my page I will only summarize the most important properties of this protocol.

The RC6 feature,

- Different modes of operation, depending on the intended use
- Dedicated Philips modes and OEM modes
- Variable command length, depending on the operation mode
- Bi-phase coding (aka Manchester coding)
- Carrier frequency of 36kHz
- Manufacturer Philips

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

Data is modulated using Manchester coding. This means that each bit (or symbol) will have both a mark and space in the output signal. If the symbol is a "1" the first half of the bit time is a mark and the second half is a space. If the symbol is a "0" the first half of the bit time is a space and the second half is a mark. This is the opposite of the RC-5 protocol!

The main timing unit is 1t, which is 16 times the carrier period
(1/36k * 16 = 444µs).

With RC-6 a total of 5 different symbols are defined:

1. The leader pulse, which has a mark time of 6t (2.666ms) and a space time of 2t (0.889ms). This leader pulse is normally used to set the gain of the IR receiver unit.

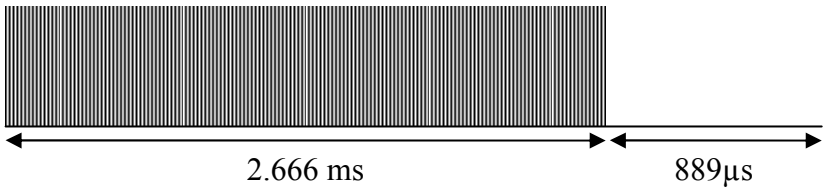


Figure 3.16 Leader pulse

2. Normal bits, which have a mark time of 1t (0.444ms) and space time of 1t (0.444ms). A "0" and "1" are encoded by the position of the mark and space in the bit time.

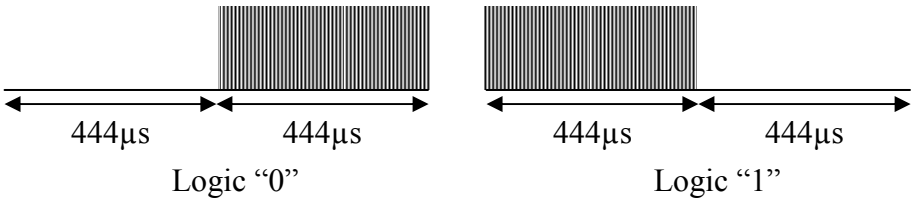


Figure 3.17 Normal bit abbreviations

3. Trailer bits, which have a mark time of 2t (0.889ms) and a space time of 4t (0.889ms). Again a "0" and "1" are encoded by the position of the mark and space in the bit time.

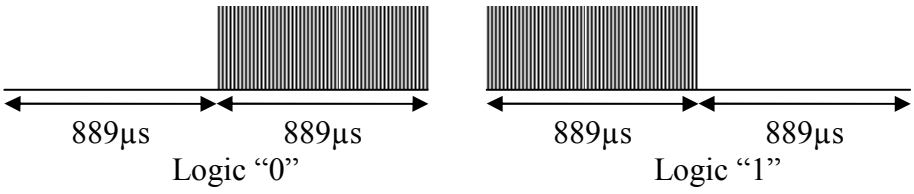


Figure 3.18 Trailer bit abbreviations

The leader and trailer symbols are only used in the header field of the messages, which will be explained in more detail below.

3.8.1 RC6 Mode 0

Mode 0 is a dedicated Philips Consumer Electronics mode. It allows control of up to 256 independent devices, with a total of 256 commands per device. The command is a concatenation of different information. I will cover these different components from left to right. (SB-Project, n.d.)

<table>
<tr><td>LS</td><td></td><td>SB</td><td>mb2 . . . mb0</td><td>TR</td><td>a7 . . . a0</td><td>c7 . . . c0</td><td rowspan="2">Signal Free</td></tr>
<tr><td colspan="5">Header</td><td>Control</td><td>Information</td></tr>
</table>

Figure 3.19 RCMM Mode 0 Packet Structure

Header field

The Header field consists of 3 different components.

- First the leader symbol LS is transmitted. Its purpose is to adjust the gain of the IR receiving unit.
- This leader symbol is followed by a start bit SB which always has the value "1". Its purpose is to calibrate the receiver's timing.
- The mode bits mb2 ... mb0 determine the mode, which is 0 in this case, thus all three bits will be "0".
- Finally the header is terminated by the trailer bit TR. Please note that the bit time of this symbol is twice as long as normal bits! This bit also serves as the traditional toggle bit, which will be inverted whenever a key is released. This allows the receiver to distinguish between a new key or a repeated key.

Control Field

This field holds 8 bits which are used as address byte. This means that a total of 256 different devices can be controlled using mode 0 of RC-6. The msb is transmitted first.

Information Field

The information field holds 8 bits which are used as command byte. This means that each device can have up to 256 different commands. The msb is transmitted first.

Signal Free Time

The Signal Free time is a period in which no data may be transmitted (by any device). It is important for the receiver to detect the signal free time at the end of a message to avoid incorrect reception. The signal free time is set to 6t, which is 2.666ms.

CHAPTER FOUR

INTRODUCTION TO NEW CONCEPT: SMART ADAPTIVE REMOTE CONTROLLER (SARC)

4.1 Introduction

As mentioned before, a new remote controller structure is started to be defined and explained in this chapter. Actually the protocol is only a tool to make this new concept usable. This concept is based on drawing keys on a touch screen which are usable for the controlled device's states one at a time. For example a DVD Recorder product has a remote controller with many keys defined statically and those keys are always on that remote controller whether they are used or not on any particular state. In this concept keys that can be used in current state are drawn on a touch screen dynamically.

4.2 Smart Adaptive Remote Controller Concept

The remote controller only contains keys to switch on the controlled device while in stand-by mode. If any device is registered or if there are more than one device registered, then remote controller may contain keys to select any of the registered devices. Figure 4.1 shows that sate of the remote controller. This state or view called base state or base view.

This base view can be change by the implementers depending of their application of product, system states or system mechanisms. For that reason this view is given here only to give an idea about the remote controller while not in communication mode. While remote controller is in this view, user can only switch on the controllable device by using standby key or any key to switch on the device. In consumer electronics other keys in addition to standby key can be switch on these equipments

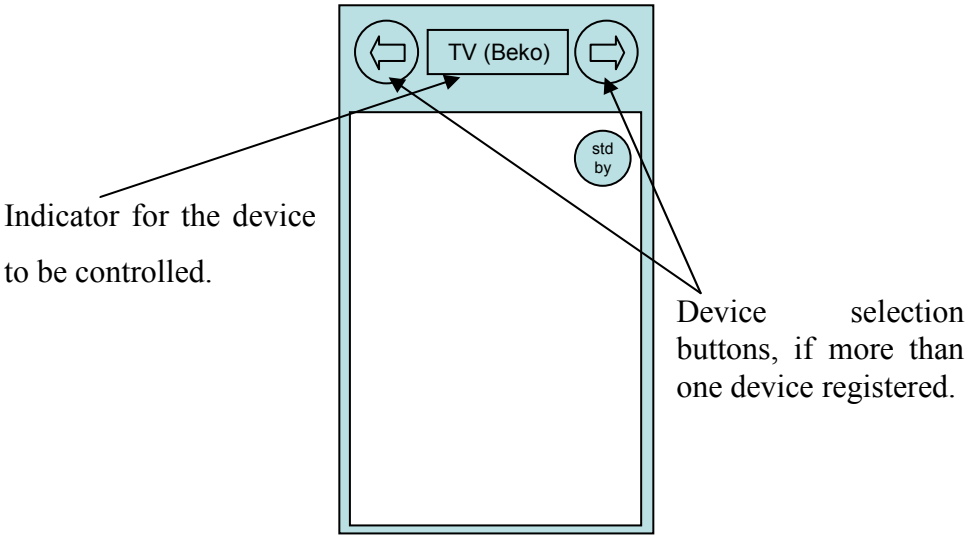


Figure 4.1 Base view of adaptive remote controller

Figure 4.1 shows the remote controller while it contains only key in order to switch on the device. That key is saved the remote controllers database when the device is registered or sent by controllable device while it is entering standby mode.

After device switched-on and complete its boot-up sequence, it sends its keys which are used in its first state reached after booting. For example one of the more complex devices on the market is DVD Recorder now. After booting this DVD Recorder device user can change the channel by using numeric keys, change the setup settings or change the target/source storage for playing or recording or record the current channel audio and video or play something from selected storage media etc. Figure 4.2 shows the DVD Recorder's keys after booting at the first state, and shows the mechanism of the new concept of remote controller messaging structure after pressing a key.

Figure 4.2 can be different depending on the devices and their application software. In Figure 4.2 user presses the HDD/DVD button to select the storage device by toggling between HDD and DVD. After pressing that button the device sends new key or keys to remote controller if pressed key changes the device's state. Here there is no key send on that state change.

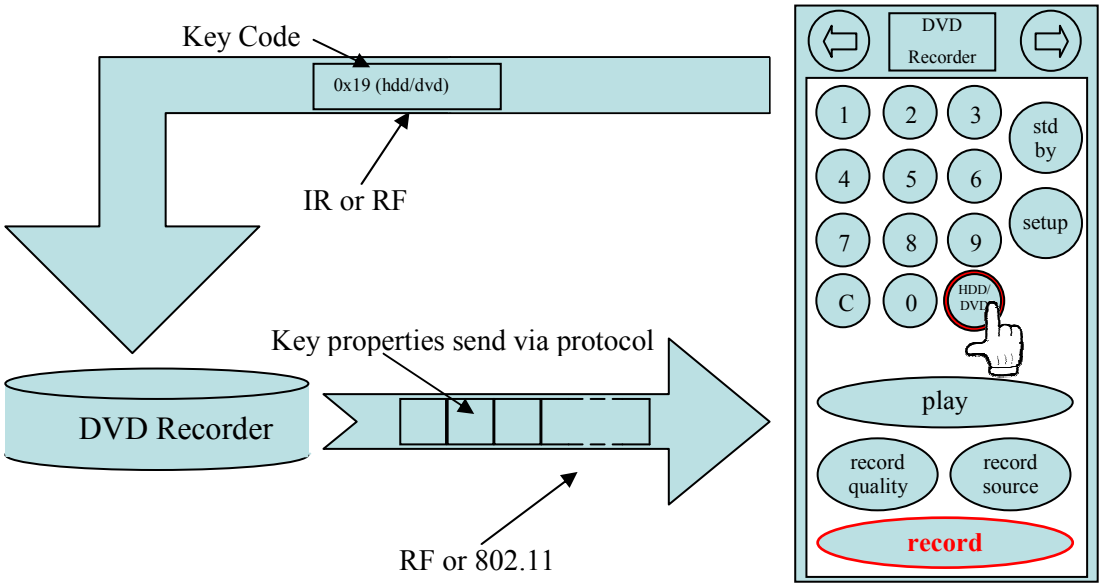


Figure 4.2 An example of state change after booting a DVD Recorder

Another example depending on Figure 4.3 if the user presses “play” key, DVD Recorder absolutely changes it is state from the current state, sends new keys and remote controller layout to the remote controller that is usable on that new state.

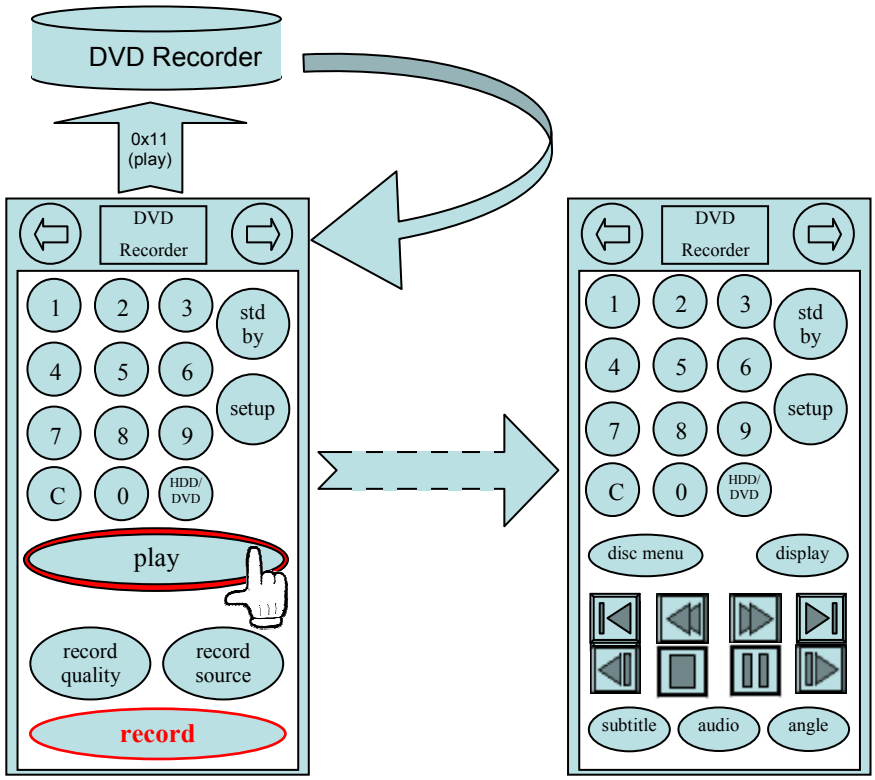


Figure 4.3 An example of state change after pressed “play” key of DVD Recorder

As understood from the above figures the concept is based on sending usable keys or remote controller layout to the remote controller while controllable device changes its state. By this, remote controller contains keys only which are useable on controllable device's states and users can be use only convenient or usable keys. Thus users will not care about which keys are convenient, usable or which keys may not create application wise problem. In case of manufacturers they do not care much about the users pressing an inconvenient key which may cause the system to enter unstable state at any state change of the device. For those reasons this remote controller approach provides good opportunities for both users and manufacturers. For users they see usable keys only or the most used of them, for manufacturers system test efforts and number of system crashes will decrease due to the absence of remote controllers unusable keys while the device's state changes.

CHAPTER FIVE

THE PROTOCOL DESIGN: SYSTEM STATES AND MESSAGING SRUCTURES

5.1 System States

The new concept system has a communication in two directions between remote controller and controllable device. Thus, new remote controllers are designed to control more than one device like a universal remote controller by using this system. One of the main purposes of this Smart Adaptive Remote Control Protocol does not need to know any controllable device or any hard coded device ID or key code of any devices. Because any suitable device or remote controller must be communicate with each other for controlling whether they know each other or not. For that reason there must be a handshake state or registration state for both in order to see each other. For remote controller, this state is learning state to learn controllable device. For controllable device, this state is learning state its remote controller. Actually the device's learning state is not important so much. But remote controller's learning state is very important. Because every these type remote controllers send its ID to the controllable device. By this controllable devices send their keys or layout structures to the remote controllers, depending on their state changes. But how can remote controllers know controllable device's keys at any time if that device does not send any key. Because of this the remote controllers learning state is more important than controllable devices learning state.

5.2 Smart Adaptive Remote Control State Diagram

The Figure 5.1 shows the remote controllers learning state (in the area closed with dashed line) and general working state diagram.

Figure 5.1 shows the learning phase. This phase starts after the remote controller is switched on. In that phase, the remote controller checks any devices that are registered or not or there is any users demand for device register. Understanding no registered device found after a result of registered devices control operation, the remote controller starts multicasting if there is any device available in order to register its database. This is performed by a timeout to avoid remote controller goes into an infinite loop.

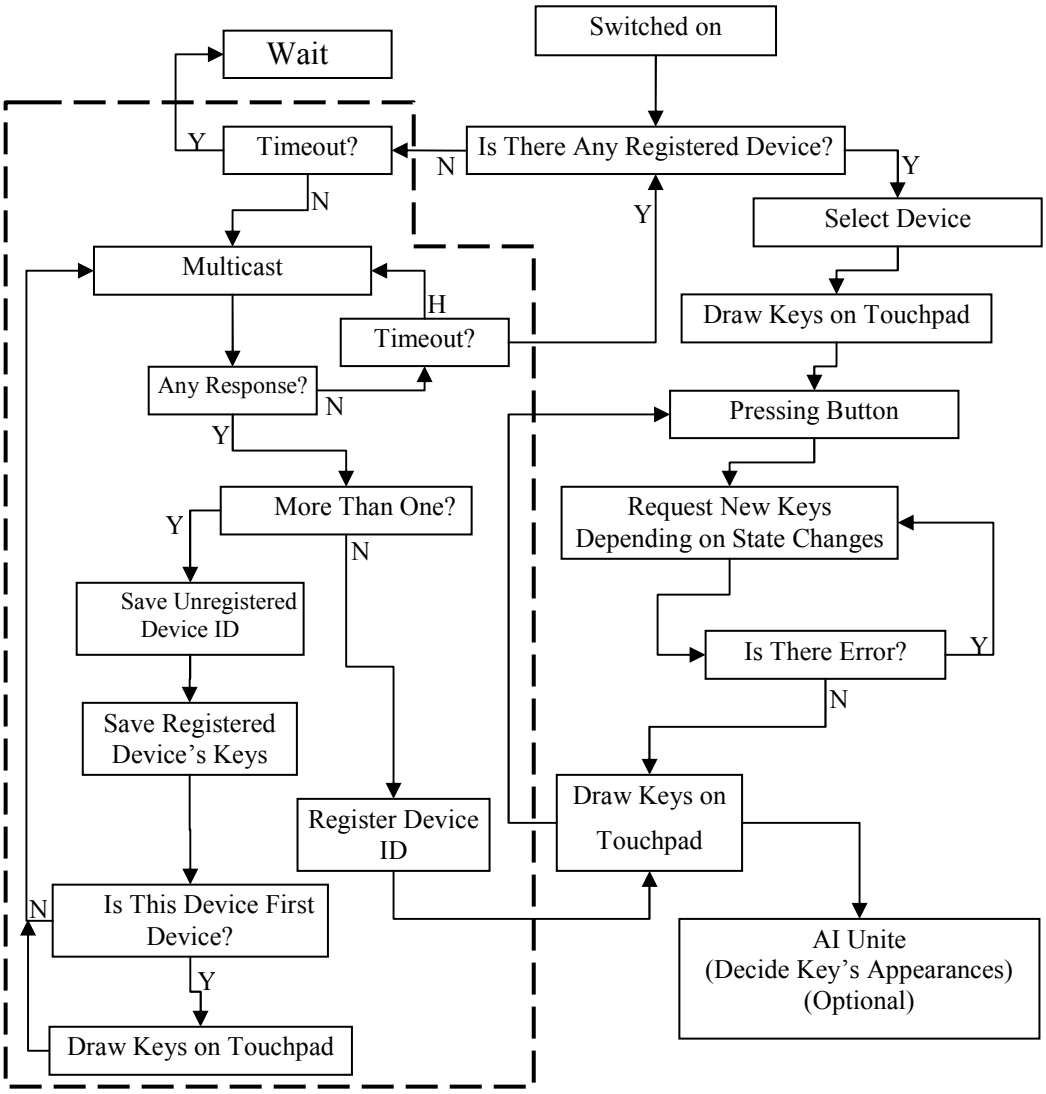


Figure 5.1 State diagram for remote controller

Other remaining part of the state diagram's in Figure 5.1 is the normal running state of the Smart Adaptive Remote Control. This normal running state starts with

checking if any registered device is available to control or not. If there is a registered device in order to control and if it is selected, then draws the keys on touchpad and waits the user to press a key. When the key is activated by the user remote controller sends it to controllable device and waits those returned keys depending on that device's state changes. If any new keys received without error from the controllable device, remote controller draws them on the touchpad and waits for a new key activation from the user. If an error occurs during receiving new keys this receiving phase request must be repeated for that keys.

5.3 Controllable Device State Diagram

In previous sections controllable device learning state is described. In this section this learning state and general running state of controllable devices are described which are controlled by Smart Adaptive Remote Control Protocol.

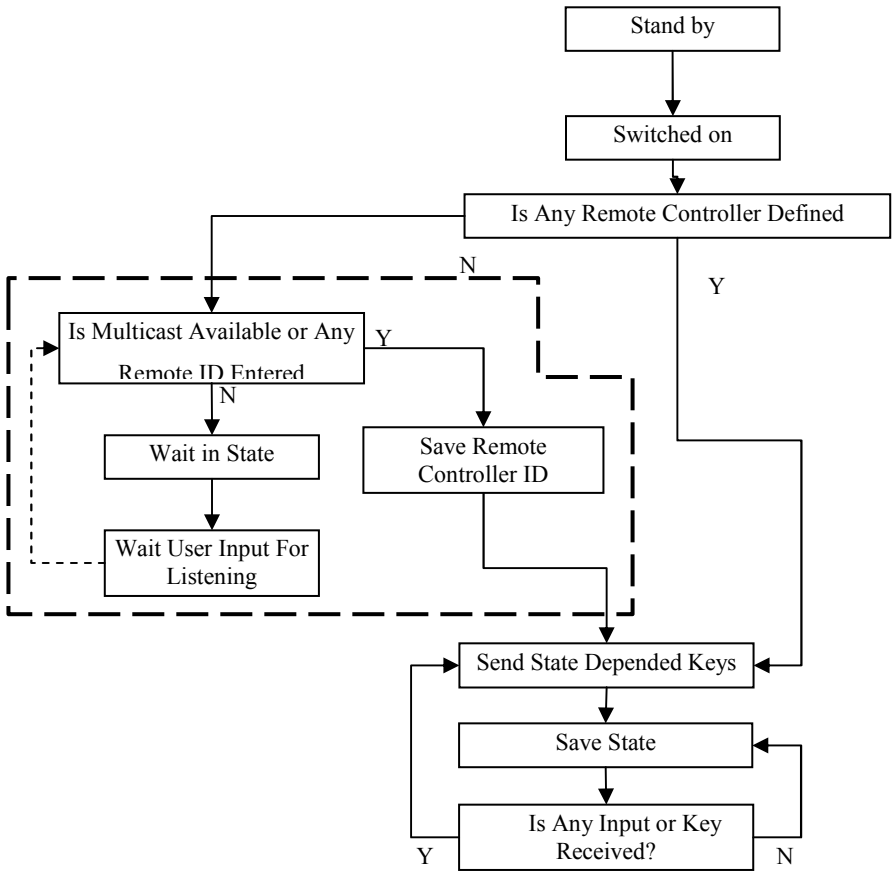


Figure 5.2 State diagram for controllable device

In Figure 5.2 the area, which is closed with dashed line, is the controllable devices' learning state. The learning state starts with checking if there is any registered remote controller or not. If there is no defined remote controller then system checks any available multicasting. If there is multicasting then the controllable device saves remote controller's ID which has made that multicasting. After that, the controllable device sends its keys, depending on its states, to the remote controller and than waits for keys from remote controller.

Other remaining part of the area of Figure 5.2 is normal running state of the controllable device. This normal running state can be easily understood from the state diagram in Figure 5.2. This is established on sending keys or layouts to the remote controller depending on its status.

CHAPTER SIX

SARCP PACKETS DEFINATIONS

6.1 SARCP Packet Structure and Definitions

Mainly there are two different packet groups in this protocol which mostly have same packet structure. One for remote controller to controllable device and the other one is from controllable device to remote controller. The other two main packet structures are ACK or NACK type primitive packets.

The packet sent by the remote controller to controllable device is very primitive. But others which are sent into the opposite direction is more complex. Because it has many different types payloads. All these packets and their structures are explained in this chapter.

The packets are differentiating their synch byte by means of their flows. The packet's synch byte value is 0x17 from remote controllers to controllable devices and the sync byte value is 0x19 into the opposite direction. The packet direction's flow is understood by this sync byte. The reason of using synch byte or the importance of the knowing direction is the usage of communication method that is based on multicasting. Because of this the sync byte is important in this approach.

6.1.1 Multicasting or Learning Packet of Remote Controller (RCMLP)

The multicasting packet of remote controller is used for learning or seeking controllable devices. This is the most important process of this approach. This packet is issued with multicasting method to all devices without pointing any devices. This operation can differ according to the transfer level of communication medium.

The RCMLP packet structure and its fields are shown below.

sync byte (0x17)	protocol version	packet type	packet number	remote ID	state ID	properties length	properties	reserved	CRC 32
1 byte	1 byte	1 byte	1 byte	4 bytes	2 bytes	2 bytes	variable length	1 byte	4 bytes

Figure 6.1 RCMLP Packet structure

Sync Byte **:** This byte is used to understand data flow or the direction of packet. These packets are directed from remote controller to controllable device if the sync byte equals to 0x17.

Protocol Version **:** This field describes the protocol version that is used in this communication.

Packet Type **:** Packet type describes the packet itself which help to understand the packet structure while packet parsing.

Packet Number **:** Packet number describes packets if they are fragmented before transmitted. This will be used if the network medium permits small sized carrier packet. In order to avoid packets' damage or lost transmitter should send packets by dividing into small sizes. The packet number starts with the *x* which defines the fragmented packets number and after this *x* decreases to *0* while each packet is being sent. The use of packet number is explained at the end of this chapter with an example.

Remote ID **:** This field is identifying the remote controller. This ID is saved by the controllable device and is used for all communications between these two devices. Remote ID is used by controllable device while sending data to the remote controller. The remote ID is a unique identifier for each remote controller. If one manufacturer has more than one remote

controller design, suitable with this protocol, all remote controllers must have different remote IDs.

State ID **:** State ID field defines controllable system states. In this packet this value must be set to 0x00. Thus controllable device understand this communication is an authentication or a learning state.

Properties Length **:** The properties length contains payload sizes that sent by this packet to the device.

Properties **:** Describes the remote controller's properties. We will give more details about this field in next chapter. The aim by sending this to the controllable device is to provide universality of the remote controllers interface while interacting with the user.

Reserved **:** This area is reserved for future use. But manufacturers can specifically use this area for their application usage.

CRC32 **:** CRC32 is used for checking error over all the packet content.

6.1.2 Command Packet of Remote Controller (RCCP)

This command packet is sent to remote controller to controllable device. And generally contains key code in its payload of the controllable device in order to conduct it. RCCP packet and its fields are shown at Figure 6.2.

sync byte (0x17)	protocol version	packet type	remote ID	device ID	state ID	payload length	payload	reserved	CRC32
1 byte	1 byte	1 byte	4 bytes	8 bytes	4 bytes	2 bytes	variable length	1 byte	4bytes

Figure 6.2 RCCP Packet structure

Sync Byte **:** This byte is used to understand data flow or the direction of packet. These packets are directed from remote controller to controllable device if the sync byte equals to 0x17.

Protocol Version **:** This field describes the protocol version that is used in this communication.

Packet Type **:** Packet type describes the packet itself which help to understand the packet structure while packet parsing.

Remote ID **:** Remote controller identifier. Controllable devices check this ID and make their operations if this ID is saved or belongs to authorized remote control in order to conduct it.

Device ID **:** Device ID field defines the controllable devices. This ID must be a unique identifier. Due to the multicast base of the protocol, the devices decide whether or not to response to this packet by using device ID.

State ID **:** State ID defines controllable devices' system states. It is important because controllable device performs their operations by this state ID and key code that sent in payload.

Payload Length **:** The payload length contains payload sizes that sent by this packet to the device.

Payload **:** The payload contains mostly the key code for controllable devices to control or conduct them. But this field must be a pointer. In the future, to extend protocol or to use different application this field may contains variable length data.

Reserved **:** This area is reserved for future use. But manufacturers can specifically use this area for their application usage.

CRC32 **:** CRC32 used for checking error over all packet content.

6.1.3 Transmission Status Packet of Remote Controller (RCTS)

The Transmission Status packet is used to understand that whether transmission is done successfully or not. RCTS packet and its field are shown at Figure 6.3.

sync byte (0x17)	protocol version	packet type	remote ID	device ID	state ID	ACK / NACK	reserved	CRC32
1 byte	1 byte	1 byte	4 bytes	8 bytes	4 bytes	1bytes	1 byte	4bytes

Figure 6.3 RCTS Packet structure

Sync Byte **:** This byte is used to understand data flow or the direction of packet. These packets are directed from remote controller to controllable device if the sync byte equals to 0x17.

Protocol Version **:** This field describes the protocol version that is used in this communication.

Packet Type **:** Packet type describes the packet itself which help to understand the packet structure while packet parsing.

Remote ID **:** Remote controller identifier. Controllable devices check this ID and make their operations if this ID is saved or the authorized remote control to conduct it.

Device ID **:** Device ID field defines the controllable devices. This ID must be a unique identifier. Due to the multicast base of the protocol, the devices decide whether or not to response to this packet by using device ID.

State ID **:** State ID defines controllable devices' system states. It is important because controllable device performs their operations by this state ID and key code that sent in payload.

ACK / NACK **:** Shows the transmissions status. If this field is ACK (means that true) than transmission performed without any error if this field is NACK (means that false) than transmission has error.

Reserved **:** This area is reserved for future use. But manufacturers can specifically use this area for their application usage.

CRC32 **:** CRC32 used for checking error over all packet content.

6.1.4 Identification Packet of Controllable Device (CDIP)

The Identification packet sent from controllable device to remote controller. The aim of this packet is informing the remote controller about itself. The controllable device sends its device ID and device name. This packet is sent only one time while learning procedure is running at the beginning of the communication. CDTS packet and its fields are shown at Figure 6.4

sync byte (0x19)	protocol version	packet type	device ID	device name	reserved	CRC32
1 byte	1 byte	1 byte	4 bytes	variable length	1 byte	4 bytes

Figure 6.4 CDIP Packet structure

Sync Byte **:** This byte is used to understand data flow or the direction of packet. These packets are directed from remote controller to controllable device if the sync byte equals to 0x17.

Protocol Version **:** This describes the protocol version that is used this communication.

Packet Type **:** Packet type describes the packet itself which help to understand the packet structure while packet parsing.

Device ID **:** Device ID field defines the controllable devices. This ID must be a unique identifier. Due to the multicast base of the protocol, the devices decide whether or not to response to this packet by using device ID.

Device Name **:** Describes the device name in order to present to the user on the remote controller.

Reserved **:** This area is reserved for future use. But manufacturers can specifically use this area for their application usage.

CRC32 **:** CRC32 used for checking error over all packet content.

6.1.5 Data Packet of Controllable Device (CDDP)

The Data packet sent from controllable device to remote controller. This packet is the main packet of this flow. The data packet can contains two different areas from packets that sent remote controller to controllable device. These fields are payload type and 2 bytes reserved area field for future use.

sync byte (0x19)	protocol version	packet		remote ID	device ID	state ID	payload		payload	reserved	CRC32
		type	number				type	length			
1 byte	1 byte	1 byte	1 byte	4 bytes	8 bytes	4 bytes	1 byte	1byte	variable length	1 byte	4bytes

Figure 6.5 CDDP Packet structure

Sync Byte **:** This byte is used to understand data flow or the direction of packet. These packets are directed from remote controller to controllable device if the sync byte equals to 0x17.

Protocol Version **:** This describes the protocol version that is used this communication.

Packet Type **:** Packet type describes the packet itself which help to understand the packet structure while packet parsing.

Packet Number **:** Packet number describes packets if packet fragmented before transmitted. This will be used if the network medium permits small sized carrier packet. In order to avoid packet damages transmitter side may send packets by dividing small size. The packet number starts the *x* that the fragmented packets count and after sending predecessor this number decreased to *0*. The use of packet number is explained end of this chapter by examples.

Remote ID **:** Remote controller identifier. Controllable devices check this ID and make their operations if this ID is saved or the authorized remote control to conduct it.

Device ID **:** Device ID field defines the controllable devices. This ID must be a unique identifier. Due to the multicast base of the protocol, the devices decide whether or not to response to this packet by using device ID.

State ID **:** State ID defines controllable devices' system states. It is important because controllable device performs their operations by this state ID and key code that sent in payload.

Payload Type **:** Describe the payload content. This will change respect of application. Because the payload can be differentiate by the

applications. The values that usable now and must be defined described below.

Single Key :Shows payload carries one key and its properties.

Multiple Key **:**Shows payload carries more than one key and their properties.

Information **:**Shows payload carries information from controllable devices.

Info with Button **:**Shows payload carries information and buttons from controllable devices. This is used for when the information have choices.

Payload Length **:** Contains the payload length.

Payload **:** The payload contains single or multiple key descriptions, information or any application data depends on payload type. Manufacturers can carry many different type data with payload that defined with payload type.

Reserved **:** This area is reserved for future use. But manufacturers can specifically use this area for their application usage.

CRC32 **:** CRC32 used for checking error over all packet content.

6.1.6 Transmission Status Packet of Controllable Device (CDTS)

The Transmission Status packet is used to understand that whether transmission is done successfully or not. CDTS packet and its fields are shown at Figure 6.6. This packet has same structure with RCTS except sync byte value.

sync byte (0x19)	protocol version	packet type	remote ID	device ID	state ID	ACK / NACK	reserved	CRC32
1 byte	1 byte	1 byte	4 bytes	8 bytes	4 bytes	1bytes	1 bytes	4bytes

Figure 6.6 CDTS Packet structure

Sync Byte **:** This byte is used to understand data flow or the direction of packet. These packets are directed from remote controller to controllable device if the sync byte equals to 0x17.

Protocol Version **:** This describes the protocol version that is used this communication.

Packet Type **:** Packet type describes the packet itself which help to understand the packet structure while packet parsing.

Remote ID **:** Remote controller identifier. Controllable devices check this ID and make their operations if this ID is saved or the authorized remote control to conduct it.

Device ID **:** Device ID field defines the controllable devices. This ID must be a unique identifier. Due to the multicast base of the protocol, the devices decide whether or not to response to this packet by using device ID.

State ID **:** State ID defines controllable devices' system states. It is important because controllable device performs their operations by this state ID and key code that sent in payload.

ACK / NACK **:** Shows the transmissions status. If this field is ACK (means that true) than transmission performed without any error if this field is NACK (means that false) than transmission has error.

Reserved **:** This area is reserved for future use. But manufacturers can specifically use this area for their application usage.

CRC32 **:** CRC32 used for checking error over all packet content.

6.2 The Use of Packet Number Field of RCMLP and CDDP

The main purpose of packet number field of RCMLP and CDDP to describe the transmitted packet counts when their payloads are fragmented into small sized packets. The aim is avoiding the packet's data lost in slow networks. The packet number starts from *x* that the fragmented packets count and after sending predecessor this number decreased to *0*. When this number reached to *0* the receiver understand whole packet arrived and ready to retrieve its data.

The transmitter has a threshold value that constraint for external effects. For example baud rate of network or maximum transmission unite of that network. Transmitter composes payload data after that transmitter fragments this composite data chunk into the small packets respect of this threshold value. After this fragmentation transmitter adds packet header and define the packet number starting from fragment count and footer (calculated CRC32).

Assume that our data has fragmented 5 pieces. The composed data and their fragments and fragmented data packets are shown at Figure 6.7.

payload 1st piece	payload 2nd piece	payload 3rd piece	payload 4th piece	payload 5th piece

0x19	0	3	4	1011	231011	7	1	*x*	payload 1st piece	0x10ACB2
0x19	0	3	3	1011	231011	7	1	*x*	payload 2nd piece	0x10AF23
0x19	0	3	2	1011	231011	7	1	*x*	payload 3rd piece	0x10DA79
0x19	0	3	1	1011	231011	7	1	*x*	payload 4th piece	0x16F923
0x19	0	3	0	1011	231011	7	1	*x*	payload 5th piece	0x1BAA79

Figure 6.7 Packet fragmentation and use of packet number field

CHAPTER SEVEN

SARCP ENTITY PROPERTIES AND THEIR ELEMENT TYPES

7.1 Types that Used in Entity Properties

Types define the some entity properties of predefined values. Some times these types are defined as enumerators and these types are explained below.

7.1.1 Version

The version describes the protocol version. In this thesis we describe only protocol structure that suitable for new remote controller approach. For that reason this protocol is only small set of may be “perfect” protocol for Smart Adaptive Remote Controller Protocol. In order to provide compatibility, we define a version area in every packet. The structure of the version is an enumerator that describes the protocol’s version. This structure can be expandable by adding new values by developing new versions of this protocol for this approach.

```
typedef enum _versions {
      VERSION_1
} versions;
```

7.1.2 Color Type

The color type describes the color values of the button aspect of their state. The state means that this button has an action that called active while user pressing or passive there is no action. The structure must be describes the button’s color value in RGB (Red, Green, Blue) mode separately. All separate values must be integer. Color propertie’s structure described below.

```
typedef _color {
      unsigned int      red;   //RED value
      unsigned int      green; //GREEN value
      unsigned int      blue;  //BLUE value
} color;
```

7.1.3 Keytype Type

The keytype type describes the key is a repeated key or not. Several keys like volume increment/decrement keys or up/down keys (depending on application) are called repeated keys. These keys are not changed the state of the controllable device. For that reason no need to sending new keys to the remote controller from controllable device, because there is no state change. The structure of the keytype is an enumerator and its values are shown below.

```
typedef enum _keyTypes {
      SINGLE,
      REPEATED
} keyTypes;
```

7.1.4 Key Group Type

The keyGroup type describes the groups that perform the operations about same type. For example for DVD playback keys that fast forward, pause, step are performing playback functionality. For that reason this keyGroup type say the remote controller key's group that must be draw on the touchpad together. The structure of this type can be defined freely with an enumerator type by each manufacturer. Thus each manufacturer defines their own groups. The remote controller understands groups by this type.

7.1.5 Key Function Type

The keyFunction type describes the differences of the key functionalities like numeric key or non numeric key. For example 0 to 9 keys are numeric keys and after pressing these key may not be cause state change on the controllable device. In order to understand this distinction between keys this keyFunction type can be used. The structure of the keyFunction type is an enumerator and its values are shown below.

```
typedef enum _keyFunctions {
      NUMERIC,
      STANDART
} keyFunction;
```

7.1.6 Screentype Type

The screentype type describes the remote controller's screen type. The structure of the screentype type is an enumerator and its values are shown below. This type's values can be expandable according to hardware's specifications.

```
typedef enum _screenTypes {
      COLOR,
      MONOCHROME
} screenTypes;
```

7.1.7 Shape Type

The shapetype type describes the key's shape. The structure of the shape type is an enumerator and its values are shown below.

```
typedef enum _shapeTypes {
      RECTANGLE,
      SQUARE,
      CIRCLE
} shapeTypes;
```

7.1.8 addSub Type

The addSub type describes the key will adding or subtracting from previous key layout. For example "play" key must be removed from key layout on the touchpad after it is pressed. This type describes with below values to say to remote controller that key will removed or added to key layout which is drawn by previous state of the controllable device. The structure of the addSub type is an enumerator and its values are shown below. Thus CDDP will not send with too much key information and network will not been so much busy by setting appropriate value of this type's.

```
typedef enum _addSubTypes {
      ATTACH,
      DEATTACH
} addSubbTypes;
```

7.1.9 payloadType Type

The payloadType describes the protocol's packets type. This type can be different application by application. But at least these are must be defined as single key, multiple key, information and information with button. This type only notifies the remote controller what payload coming and how it is interpreted. The structure of the payloadType is an enumerator and its values are shown below. These values can be expandable depending on the applications.

```
typedef enum _payloadTypes {
      SINGLE_KEY,
      MULTIPLE_KEY,
      INFORMATION,
      INFORMATION_WITH_BUTTON
} payloadType;
```

7.2 Entity Properties Used in SARCP

In SARCP and its application to provide compatibility some properties that used in SARCP must be defined. For example key properties. Because supporting universal controllability especially keys and some other properties must be defined strictly.

But these entities are elements of the presentation layer. In the protocol layer we did not need to define these entities. But there is a common library between controllers and controllable devices to present keys and other visual elements after arriving to remote controller. And the library definitions can be sending to controllable device by RCMLP packet at remote controller while it is in learning phase.

7.2.1 The Remote Controller Property

This property describes the remote controller's structure. These are remote controller's screen dimensions, resolutions, color range and widget type's etc. The aim by sending this to the controllable device is to provide universality of the remote controllers interface while interaction with the user. These properties are resolutions

(height, width), screen type (color or monochrome) and supported widget type or libraries. The entity structure is shown below.

height : An integer value that describe the height of the screen.

width : An integer value that describe the width of the screen.

screen type : An integer value that describes the screen type.

widget library : An integer value that describes the widget library of the remote controller that used for user interface drawing. This value is an enumerator type and their values are started from 1 and continue the defined widget library. The controllable devices do not have to know this value. But they must know their entire key that sent will drawn by that widget library.

The remote controller property code structure is written is below.

```
typedef _remoteController {
      int         hegiht;
      int         width;
      screenTypes screenType;
      int         widgetLibrary;
} remoteController key;
```

7.2.2 Text Property

The text property select the predefined text font and size in the remote controller depending on the widget library that supported by remote controller. This structure can be different depending of the widget library.

7.2.3 Key Property

The keys are in this system most important. Because remote controller draw them on its touch screen by their properties and system interact with users with these keys that defined by their properties. And to provide universality of remote controller

every controllable devices send their keys an understandable format to interpret by every remote controller. The key properties listed below.

key ID : The unique key identifier. Keys are known by this ID. keyID is a unique number for every device and any keys. This means that same keyID's can be possible for different devices.

height : Defines the height of the button.

width : Defines the width of the button.

radius : Defines the radius of the button.

shape : Defines the shape of the button.

passive color : Defines the color while it is drawn. Not pressed state.

active color : Defines the color while it is pressed.

effect : Defines the blinking effect on or off that button.

text : The text that written on the button after drawing.

image : Image which is drawing on the key. This image property is depending on defined widget in the remote controller. That reason I do not define this property here.

textAlignmet : The text layout on the button.

iconAlignmet : The icon layout on the button.

key type : The key type property and its values are explained above.

key group : The key group property and its values are explained above.

key function : The key function property and its values are explained above.

must : The must property indicates that key always drawn on the touch screen without asking AI unit. In some states users may not use some keys which are usable in that state. But that keys are always sends to the remote controller. If the user does not use these keys before; if available; the remote controller AI unit can hide these keys by looking their previous usage frequency and if they have not use so much before than these keys may not

drawn on the touch screen of the remote controller. This can be 1 or 0 to indicate this property.

lock after : The lock after property indicates that after the key is used the key pad on the touch screen whether locked or not. This can be 1 or 0 to indicate this property.

addSub : This area indicates the key is adding or subtracting to previous key. This will increase the throughput of the drawing system.

key code : The key code is the describe keys code that sending to the controllable device to inform for performing necessary operations.

The key type code structure is written is below.

```
typedef _key {
      int               keyID;
      int               hegiht;
      int               width;
      int               radius;
      shapeTypes        buttonShape;
      color             passiveColor;
      color             activeColor;
      boolean           effect;
      char[15]          text;
      image             icon;
      alignment         textAlignmet;
      alignment         iconAlignmet;
      keyTypes          keyType;
      keyGroups         keyGroup;
      keyFunctions      keyFunction;
      boolean           must;
      boolean           locAfter;
      addSubbTypes      keyCode;
      int               keyCode;
      } key;
```

7.2.4 Information Property

The information property can be used for sending informative messages. Any system can send informative message to the remote controller that supports this protocol whether they are controllable or not. For example a washing machine can send its state for informing its users about a problem getting water inside.

The informative messages can be plain messages or sometimes they can be having choices to give to user some decisions. These type informative messages have some buttons like windows warning messages like “yes”, “no” or “OK”. Buttons are used from standard predefined widget library. That reason there is no need to define button properties if will used. But only button code must be defined and send to the remote controller.

message **:** Describe the information message to the user.

button OK code **:** Defines the OK button code if needed.

button YES code **:** Defines the YES button code if needed.

button NO code **:** Defines the NO button code if needed.

The informative messages code structure is written is below.

```
typedef _ infoMessage {
      char[120]   message;      //message text
      int         buttonOKCode; //OK button code ID if needed
      int         buttonYESCode;//YES button code ID if needed
      int         buttonNOCode; //NO button code ID if needed
      } infoMessage;
```

The informative messages can be plain messages or sometimes they can be having choices to give to user some decisions. These type informative messages have some buttons like windows warning messages like “yes”, “no” or “OK”. Buttons are used from standard predefined widget library. That reason there is no need to define button properties if will used. But only button code must be defined and send to the remote controller.

CHAPTER EIGHT

THE SARCP SIMULATION

8.1 Introduction

In order to enable this concept, a new protocol and mechanisms are defined at the previous chapters. And in order to prove this smart adaptive remote control protocol applicable, a simulation program is developed. The protocol provides a common language to make the remote controller and the controllable devices understand each other. This chapter explains our simulation application, which platform is used to develop, how this application is created and which components are used, what is the definition of simulated device and its states. End of all, the run-time screen shots will be given.

Our application simulates two devices, one is the remote controller and the other one is controllable device as a DVD Recorder. The controllable device's state diagram will be given in section 8.4. Please note that this is a simulation about proving the smart adaptive remote control protocol if it is applicable or is working.

8.2 Development Platform

The simulation program is developed at Borland C++ Builder. Borland C++ Builder is an object-oriented, visual programming tool to develop 32-bit applications for Windows platforms. By Borland C++ Builder highly efficient programs can be created in a short time without writing thousands lines of code. C++ Builder provides a suite of Rapid Application Development (RAD) tools at design and run. Borland C++ Builder supports object-oriented programming with two extensive class libraries. These libraries are Visual Component Libraries (VCL) and Borland Component Library for Cross-Platform (CLX).

Borland C++ Builder also provides an integrated debugger for following code flow or finding and fixing errors in codes. This debugger is very powerful and practical tool for computer applications. But in real time applications this debugger is not helpful, especially if application has network communications. But it was very helpful. It shows variable values while you are trace the program line by line.

Borland C++ Builder is provides hundreds of components with respect of those two libraries. Programmers can use components to develop very powerful applications. Of course many of these components are used in our simulation application.

8.3 Application Properties

This simulation application simulates a remote controller and a controllable device with a protocol by using Ethernet communication infrastructure and its physical layer. Because there are two different devices in this simulation application. And these devices run on two different computers to provide real system behaviors. For that reason this application uses standard Ethernet network for communication.

This application placed on the IP structure at protocol stack. Actually the smart adaptive remote control protocol is independent from communication medium and it’s infra structure. It is designed to support this independency. This protocol can be run on many communications medium, i.e. RF application like Bluetooth or wireless networks by using its well defined properties.

In order to provide communication between two end points our program uses benefits of sockets. A server socket component from VCL library of Borland C++ Builder used for remote controller in order to simulate it and a client socket component from same library used for controllable device in order o simulate it. Client and server sockets are explained in the following section.

8.3.1 Sockets, Server and Client Sockets

8.3.1.1 Sockets

Sockets were created at Berkeley University in California in order to permit network communication with UNIX systems. A socket is a connection between two hosts or in the other words socket is a method which allows applications to communicate between them. Sockets perform seven basic operations depending on being server socket or client socket. These operations are;

- Connect to a remote machine
- Send data
- Receive data
- Close a connection
- Bind to a port
- Listen for incoming data
- Accept connections from remote machines on the bound port

The first four operations are used for both server and client sockets. The last three operations are needed only by servers, which wait for clients connections (Harold, 2004)

Data is transmitted across the Internet in packets of finite size called datagrams. Each datagram contains a header and a payload. The header contains the address and port to which the packet is going, the address and port from which the packet came, and various other information used to ensure reliable transmission. The payload contains the data itself. However, since datagrams have a finite length, it's often necessary to split the data across multiple packets and reassemble it at the destination. It's also possible that one or more packets may be lost or corrupted in transit and need to be retransmitted or that packets arrive out of order and need to be reordered. Keeping track of this—splitting the data into packets, generating headers, parsing the headers of incoming packets, keeping track of what packets have and haven't been received, and so on—is a lot of work and requires a lot of complicated code.

Fortunately, sockets allow the programmer to treat a network connection as just another stream onto which bytes can be written and from which bytes can be read. Sockets shield the programmer from low-level details of the network, such as error detection, packet sizes, packet retransmission, network addresses, and more.

8.3.1.2 Client Sockets

A client is the host that initiates the connection with a server by socket. As mentioned before the operations of clients socket. Many programs normally use client sockets in the following order:

1. The program creates a new socket with a constructor.
2. The socket attempts to connect to the remote host.
3. Once the connection is established, the local and remote hosts get input and output streams from the socket and use those streams to send data to each other. This connection is full-duplex; both hosts can send and receive data simultaneously. The data depends on the protocol.
4. When the transmission of data is complete, one or both sides close the connection.

Client socket is opens a connection by defined port to the server from a computer. There must be a server other side of the connection to perform a communication or data exchange. Client sockets are behaves like a requester in order to start communication with any server specified by a port. (Harold, 2004)

8.3.1.3 Server Sockets

A server socket is a listener which listens for incoming socket connections. The socket will listen on a specific port and respond when a message is sent to that port. The incoming messages may be from the same machine or another machine.

A server program that used server socket basically performs following operations;

1. A new socket is created on a particular port using a server socket constructor.

2. The server socket listens for incoming connection attempts on that port using accept method.
3. Gets input and output streams that communicate with the client.
4. The server and the client interact according to an agreed-upon protocol until it is time to close the connection.
5. The server, the client, or both close the connection.
6. The server returns to step 2 and waits for the next connection.

A server socket runs on the server and listens for incoming TCP connections. Each server socket listens on a particular port on the server machine. When a client on a remote host attempts to connect to that port, the server wakes up, negotiates the connection between the client and the server –accept this connection– and returns a regular socket descriptor representing the socket between the two hosts. In other words, server sockets wait for connections while client sockets initiate connections. Once a server socket has set up the connection, the server uses a regular socket descriptor to send data to the client. Data always travels over the regular socket. (Harold, 2004)

8.4 Simulated Device and Device States

In this simulation application there are two different devices simulated. DVD recorder is the controlled device. And remote controller is a Smart Adaptive Remote Controller. These two devices are explained in the following sections.

8.4.1 Controllable Device Application as DVD Recorder

The controllable device is a simulation of DVD Recorder. Their basic states and keys are simulated in application. Here DVD Recorder performs DVD playback, recording to DVD, time shifting, and increase and decrease the volume.

This simulated device's states can change with key commands coming from remote controller across from the network. These states diagram are given in Figure 8.1.

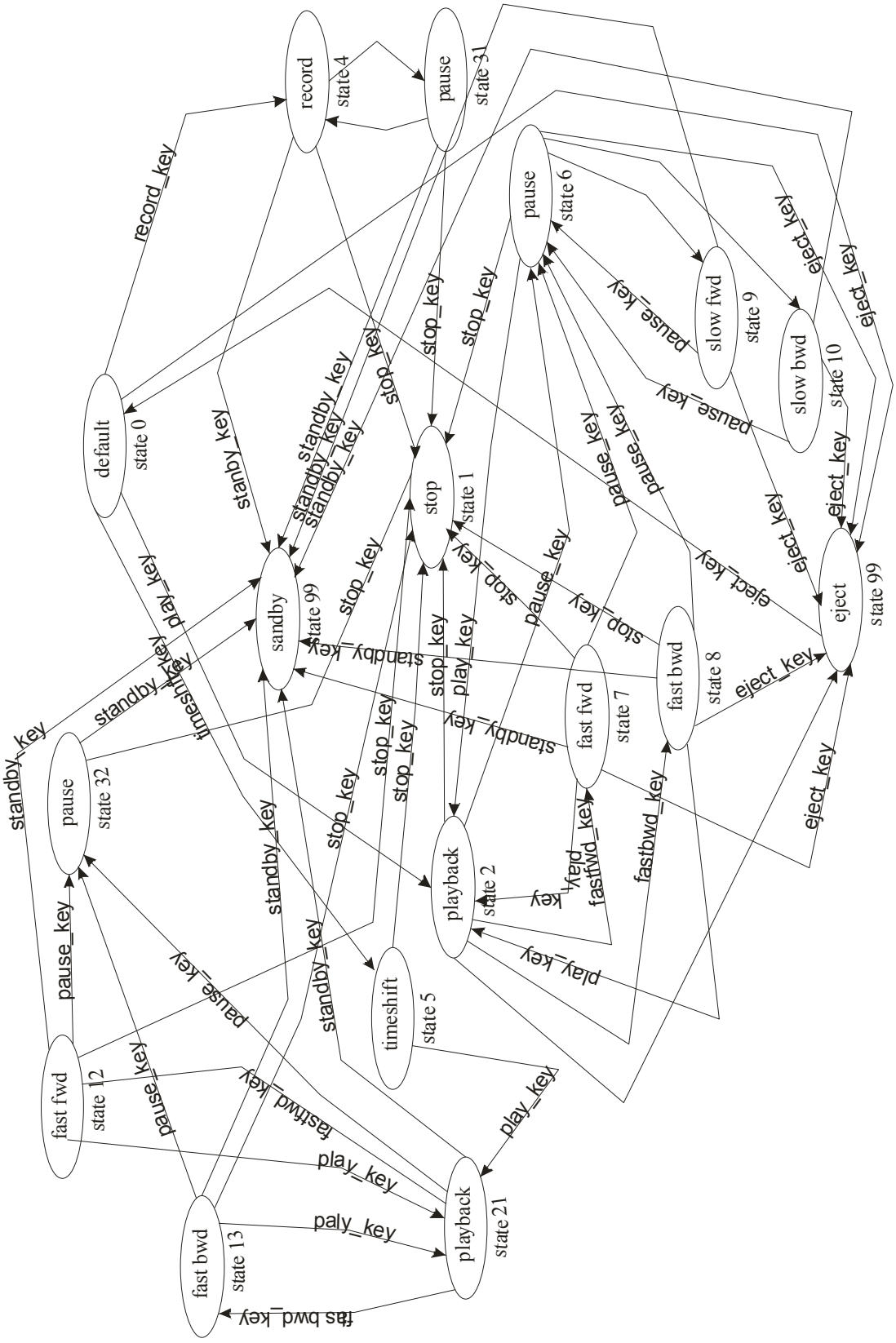


Figure 8.1 State diagram of controllable device

8.4.2 Remote Controller Application

The remote controller simulation application only performs the receiving key information, drawing these keys to screen and sending their key code to the controllable device across the network. Screenshots may give some idea about remote controller.

8.5 Application Screenshots

In this section several screenshots are given about remote controller and controllable device from simulation.

Figure 8.2 shows the remote controller is switched on and has no connections from any controllable device. Figure 8.3 shows remote controller switched on and has a connection from registered controllable device. This screenshot was captured while raw data pane was selected. Raw data pane shows you any communication data coming from the network with smart adaptive remote controller protocol. Figure 8.4 shows controllable device is switched on and connected to the registered remote controller. Figure 8.5 shows controllable device with raw data pane tab after sending first CDDP to the remote controller. Figure 8.6 shows the remote controller with protocol pane after receiving CDDP from the controllable device. Figure 8.7 shows the remote controller with payload pane after receiving a CDDP from the controllable device. Payload pane shows you payload after parsed its content. From Figure 8.2 to Figure 8.7 are about initialization of remote controller and controllable devices states, in other words their first states after they switched on.

Figure 8.8 shows remote controller after play key is pressed. After pressing play key (or any key) remote controller sends RCCP with play key command to controllable device and a new CDDP is received from controllable device which carries keys of controllable device's new state. The Figure 8.9, Figure 8.10 and Figure 8.11 show after pressing play key from the remote controller. Figures are captured from the controllable device's alternately protocol pane, payload pane and raw data pane.

Figure 8.12, Figure 8.13, Figure 8.14 show remote controller after pressed standby key. Figures are captured from remote controller alternately raw data pane, payload pane and protocol pane of remote controller's. Figure 8.15 shows controllable device after pressed standby key.

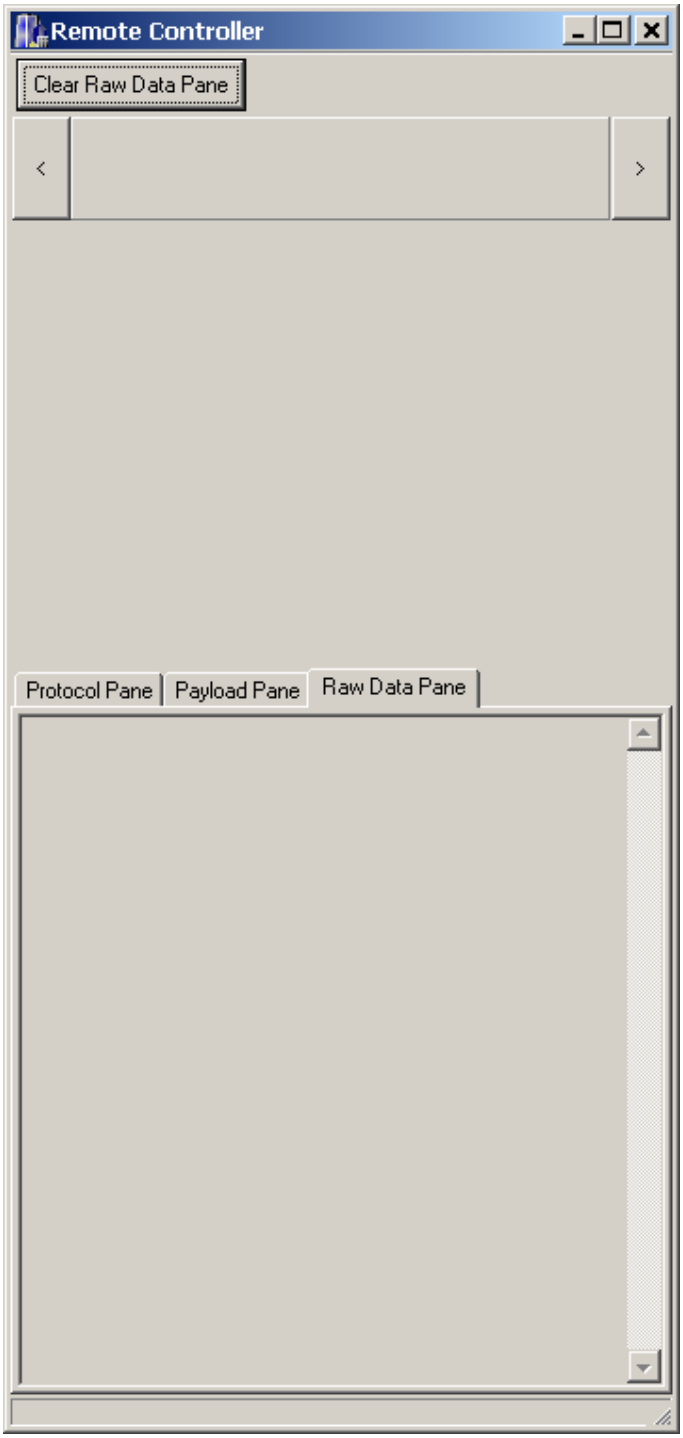


Figure 8.2 Remote Controller after switched on.

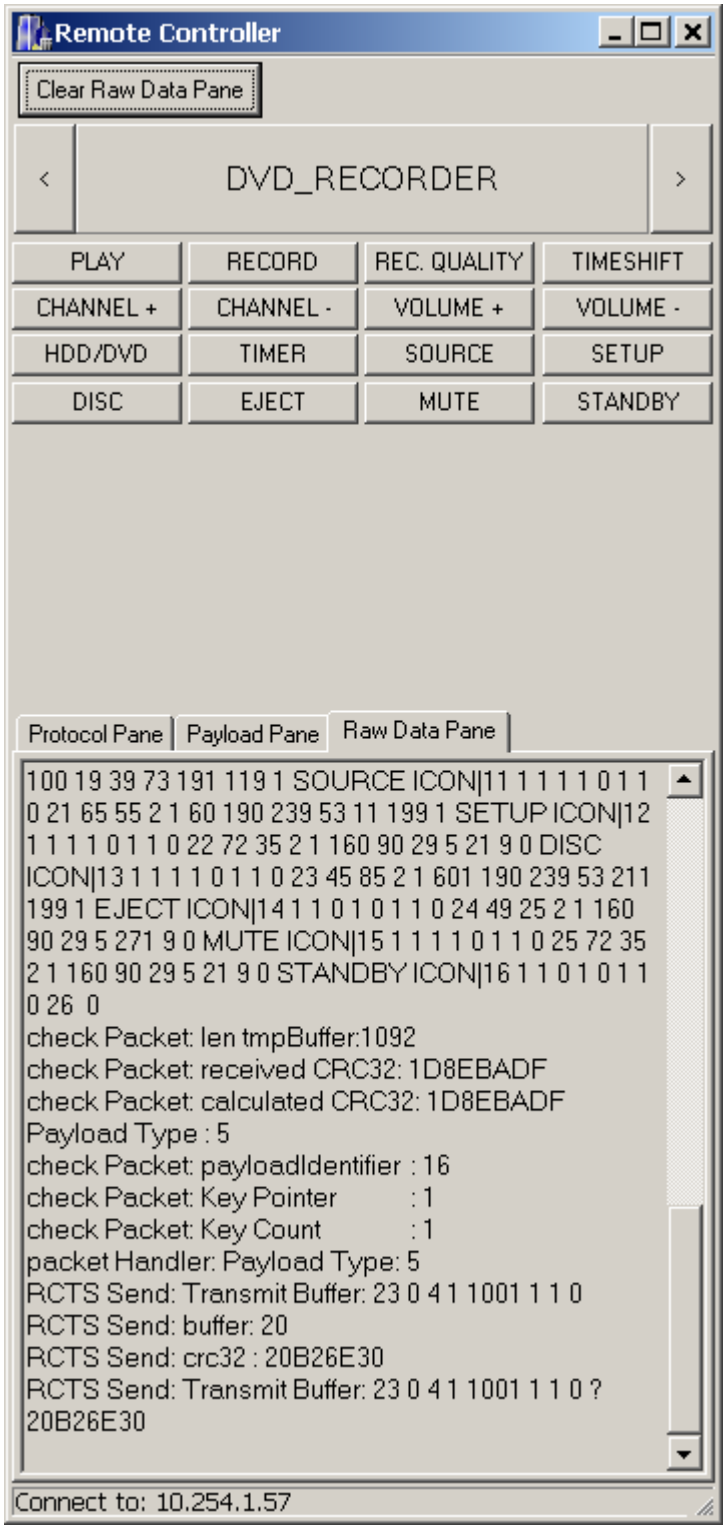


Figure 8.3 Remote Controller after receiving CDDP from controllable device.

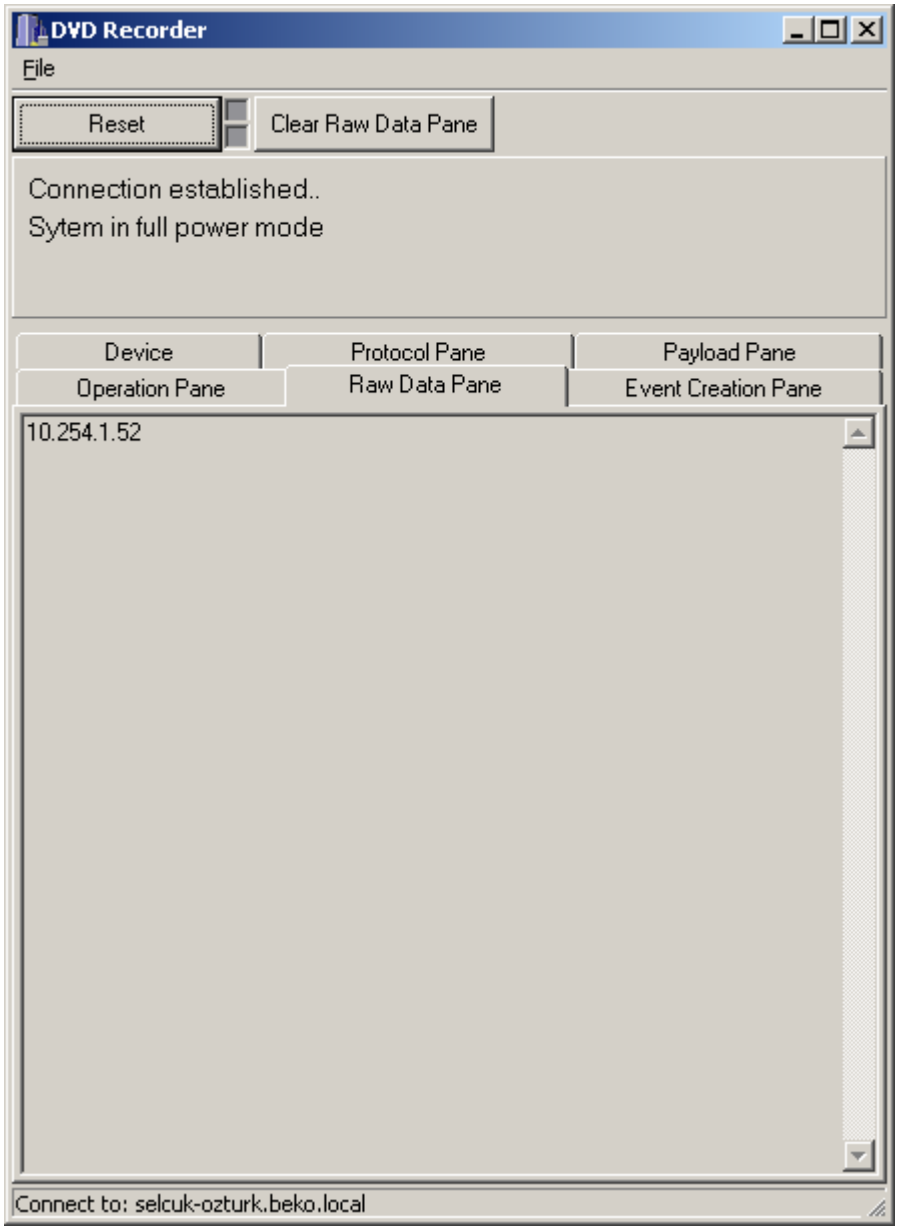


Figure 8.4 Controllable device after being connected to the registered remote controller.

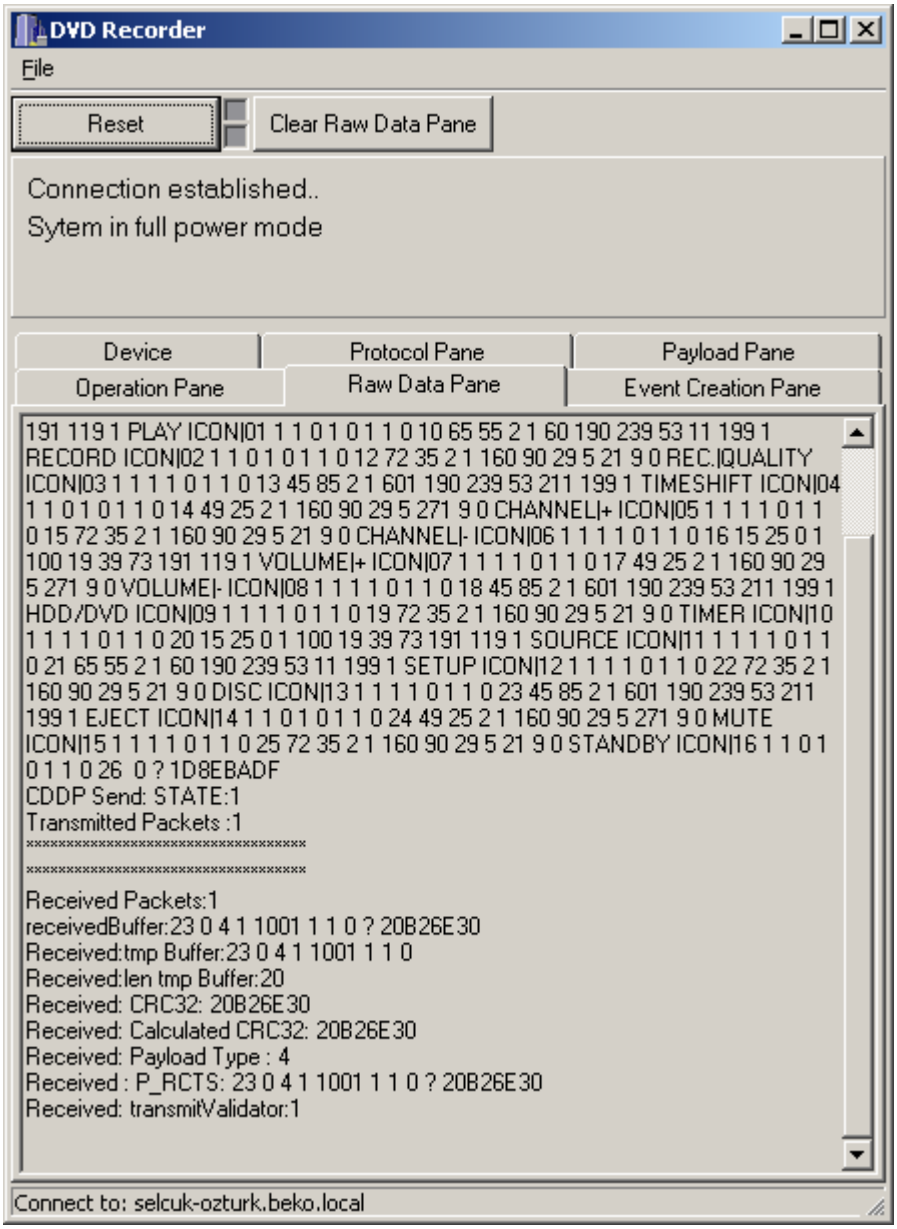


Figure 8.5 Controllable Device with raw data pane after sending first CDDP to the remote controller.

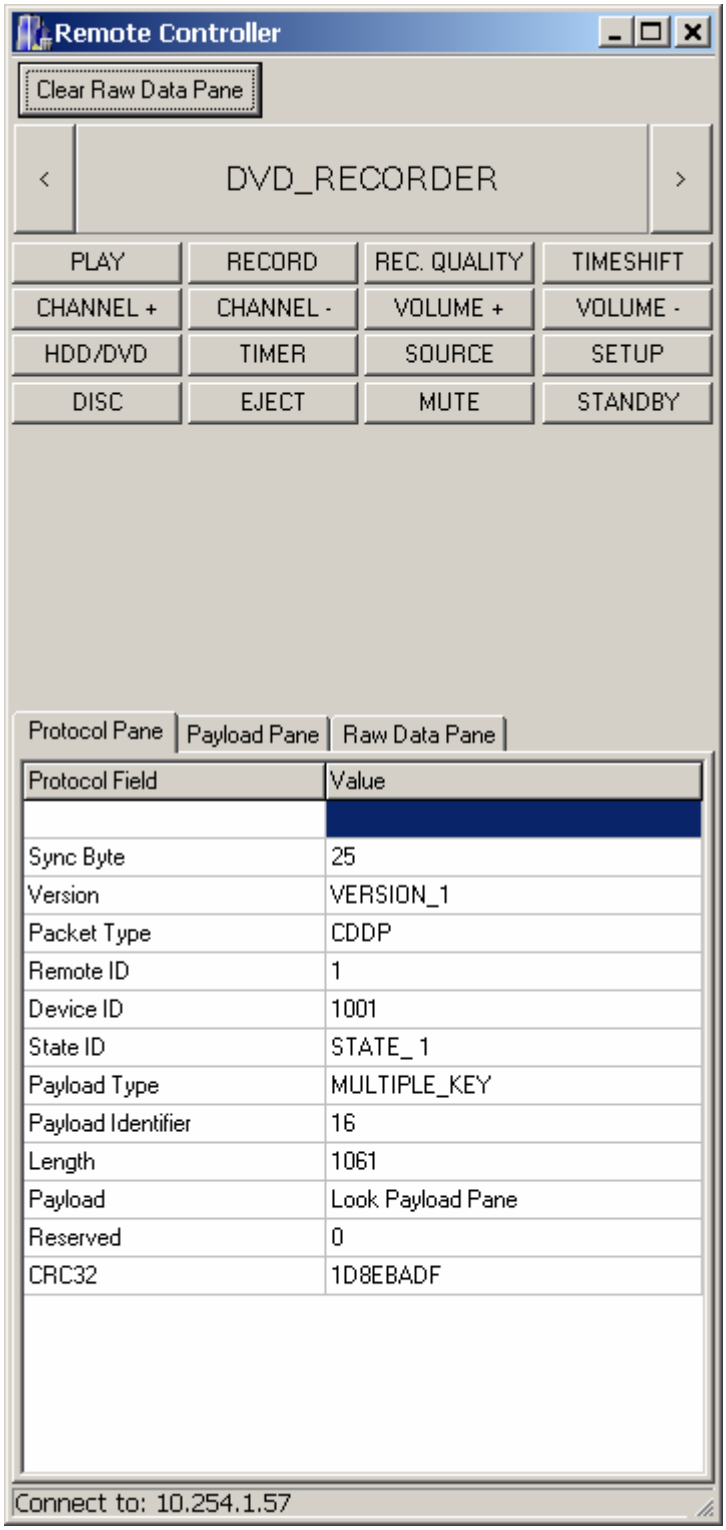


Figure 8.6 Remote Controller with protocol pane after CDDP is received.

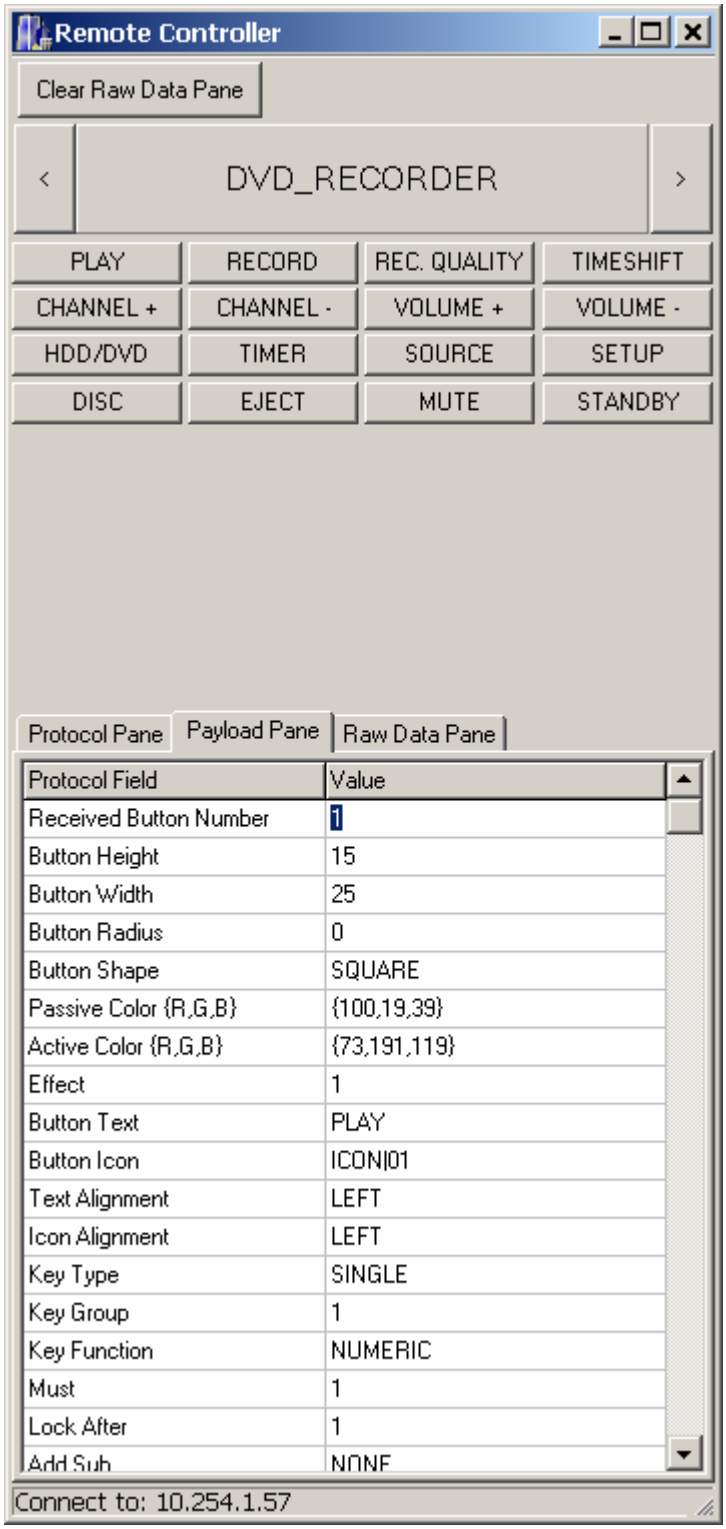


Figure 8.7 Remote Controller with payload pane after CDDP is received.

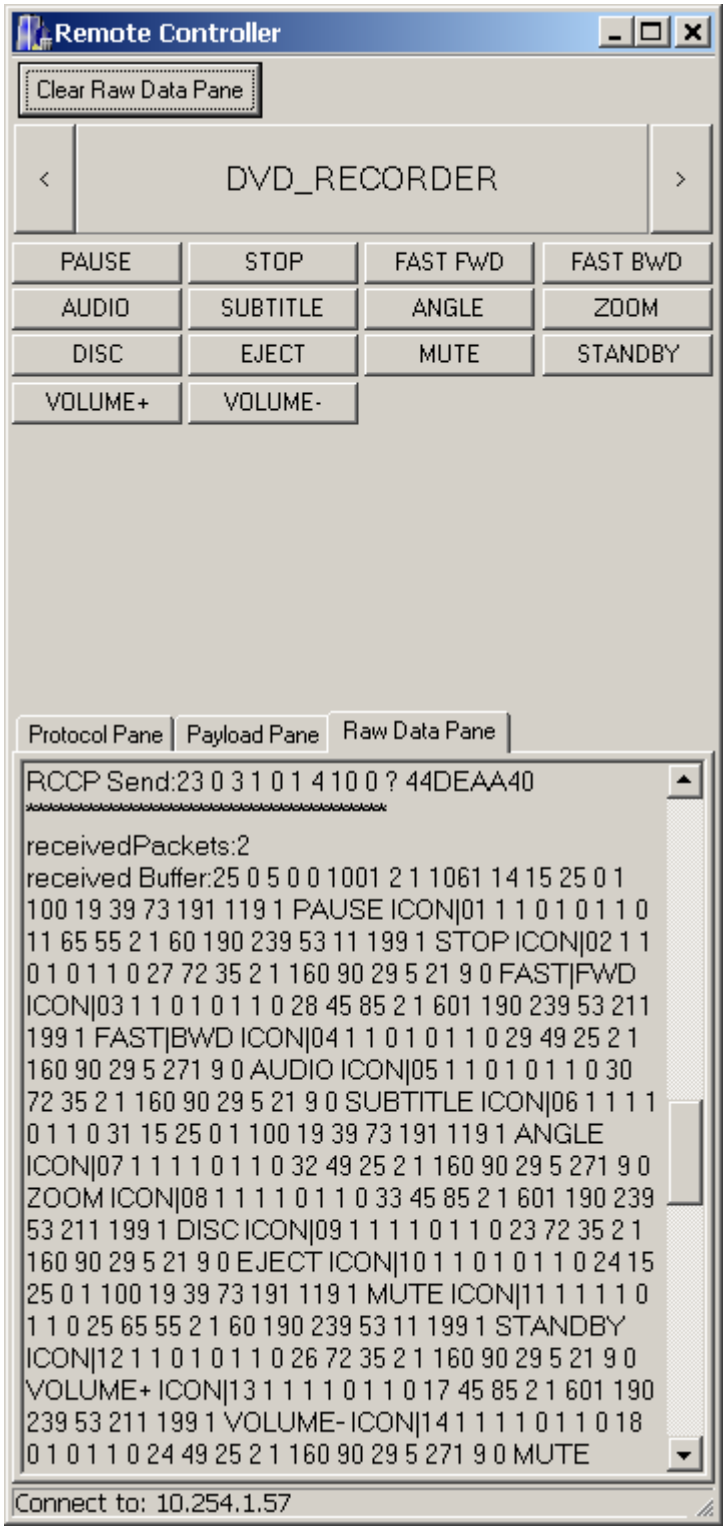


Figure 8.8 Remote Controller after CDDP is received from the controllable device.

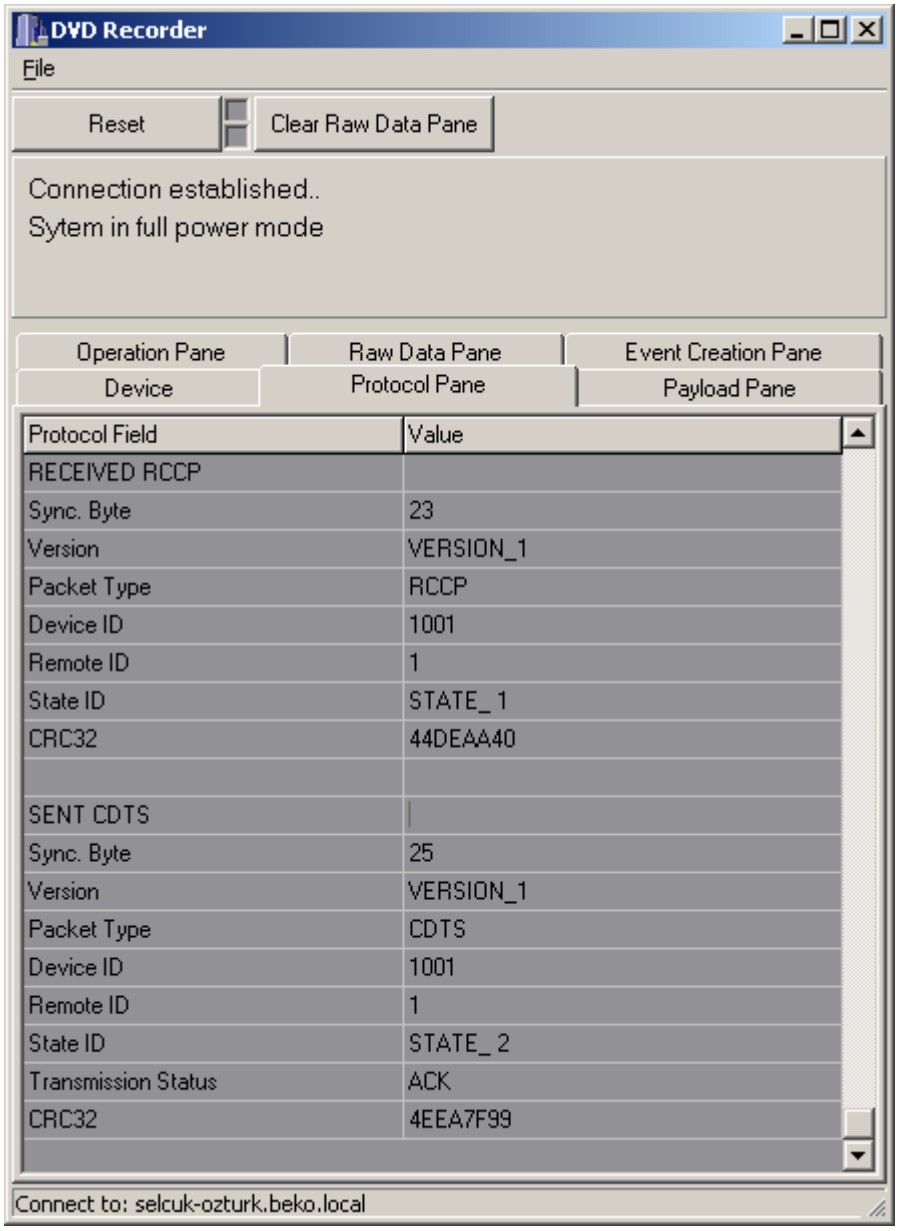


Figure 8.9 Control Devices protocol pane after pressing play key from the remote controller.

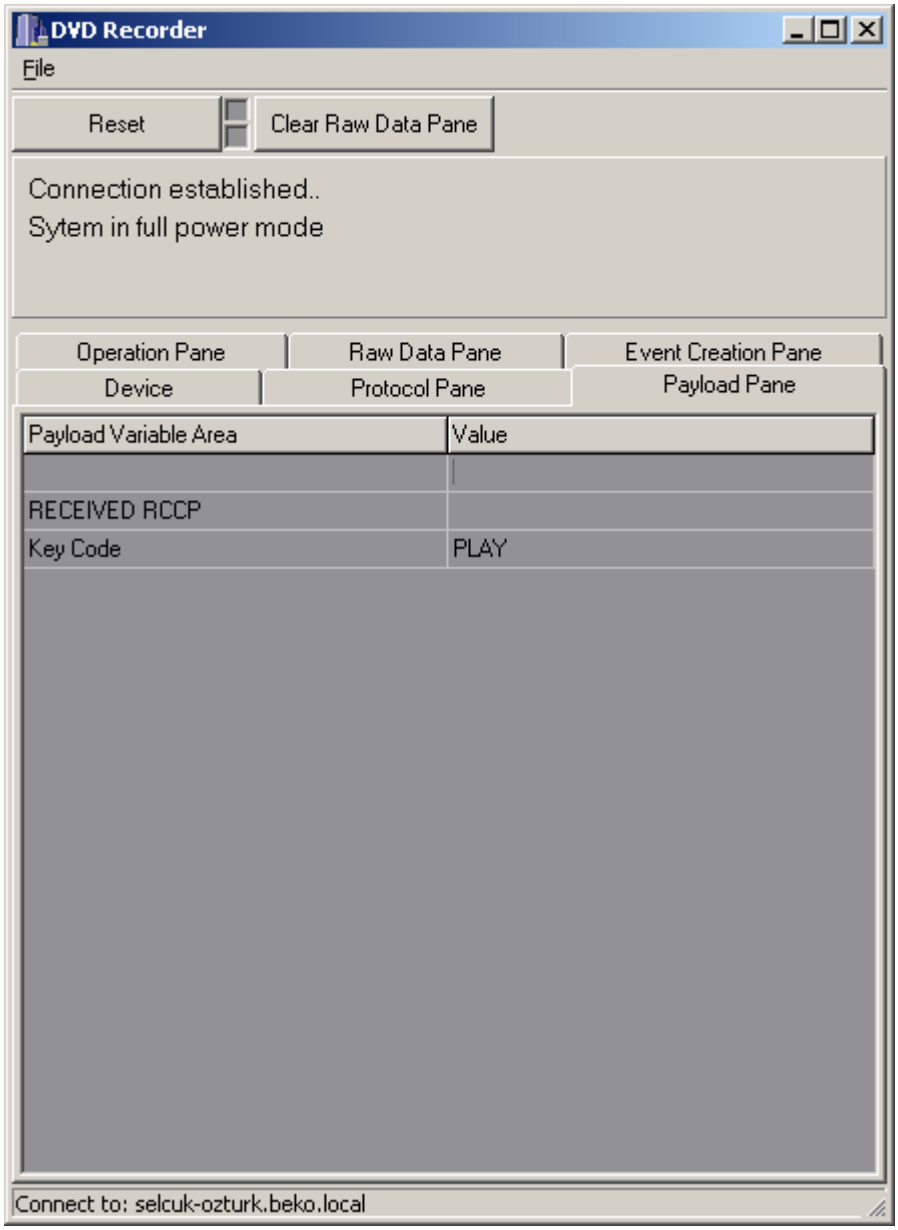


Figure 8.10 Control Devices protocol pane after pressing play key from the remote controller

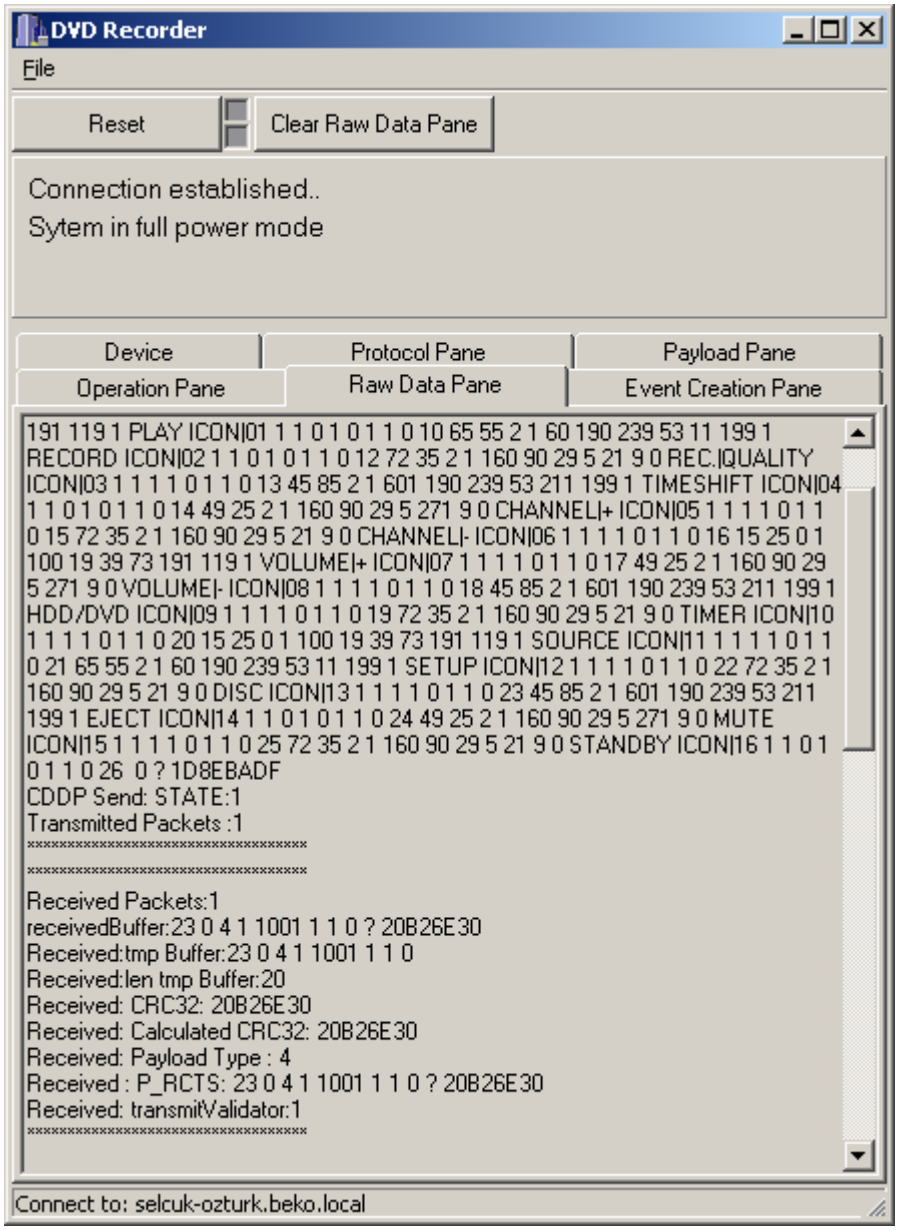


Figure 8.11 Control Devices raw data pane after pressing play key from the remote controller

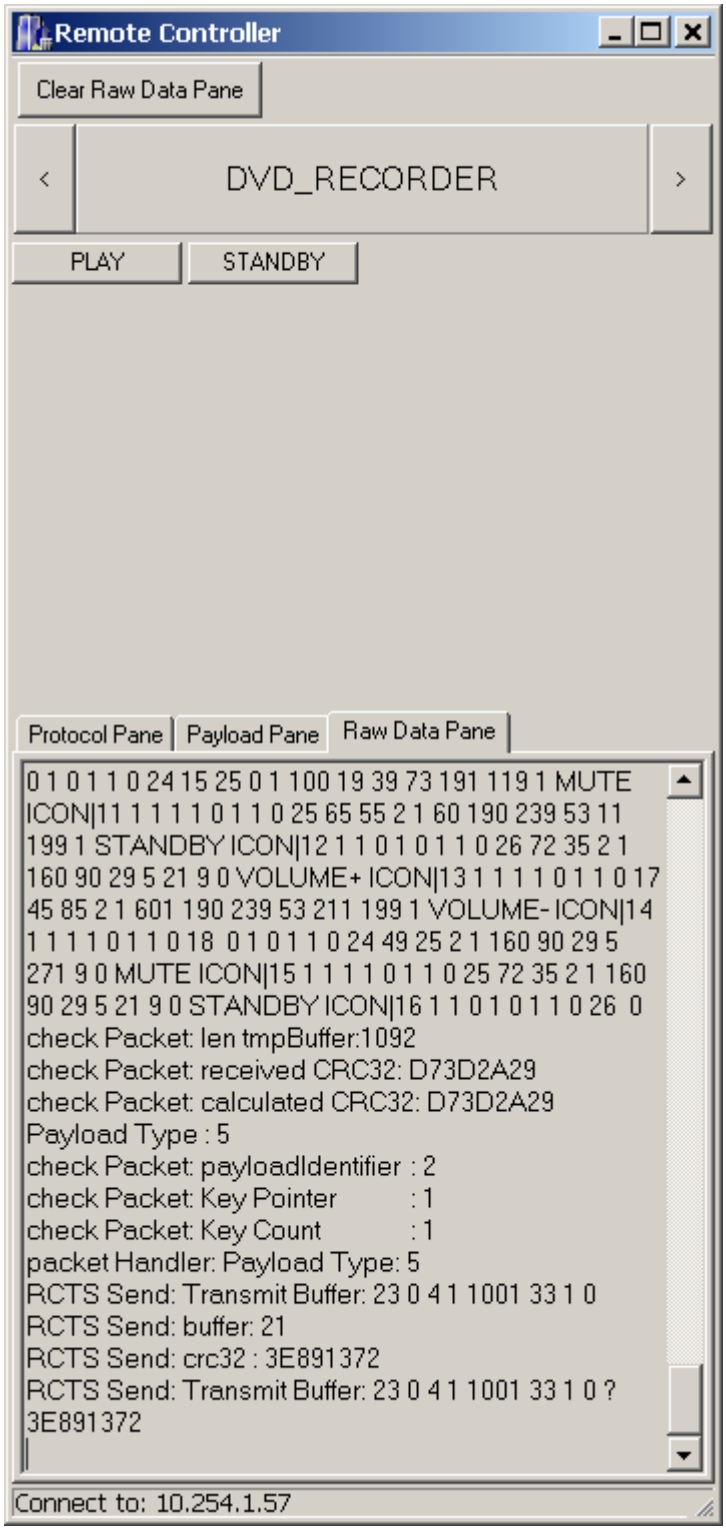


Figure 8.12 Remote Controller's raw data pane after pressing standby key

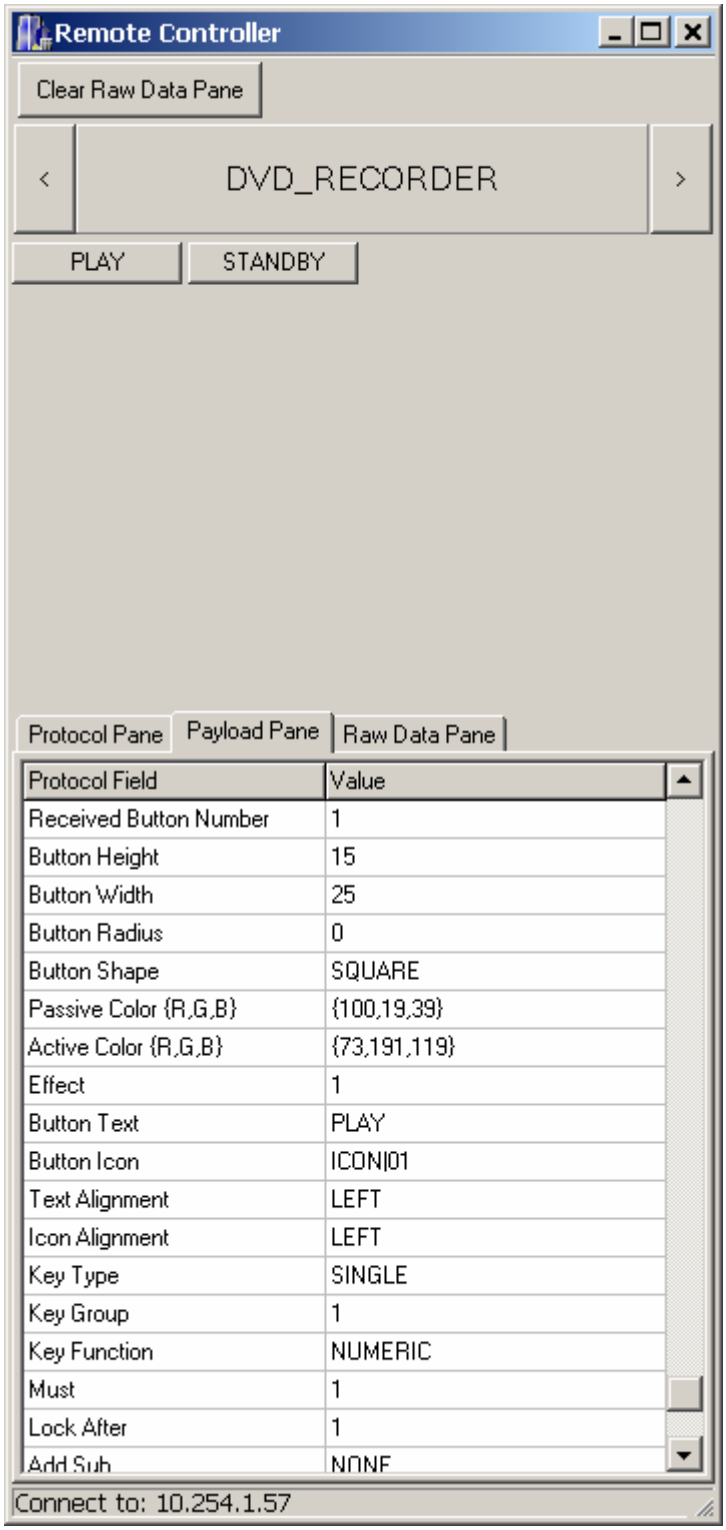


Figure 8.13 Remote Controller's payload pane after pressing standby key.

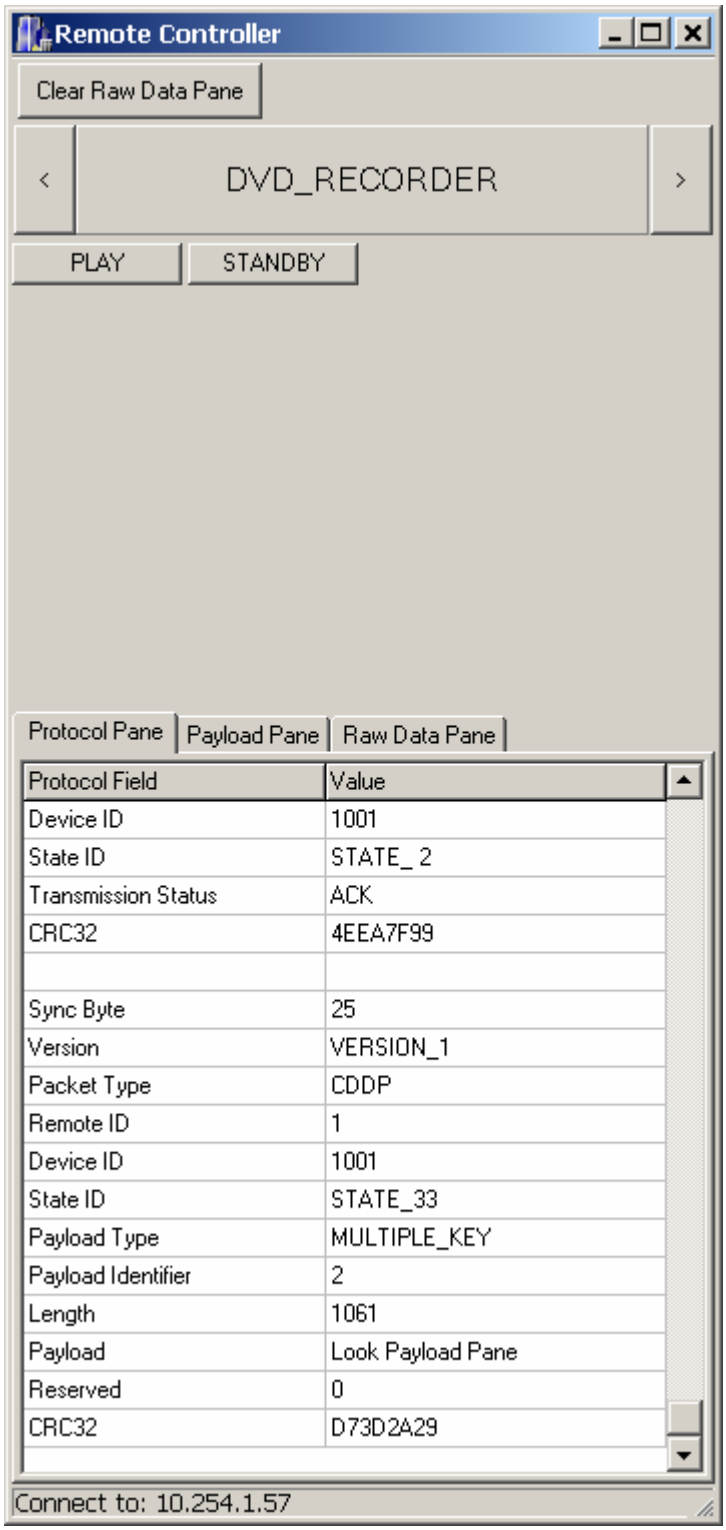


Figure 8.14 Remote Controller's protocol pane after pressing standby key.

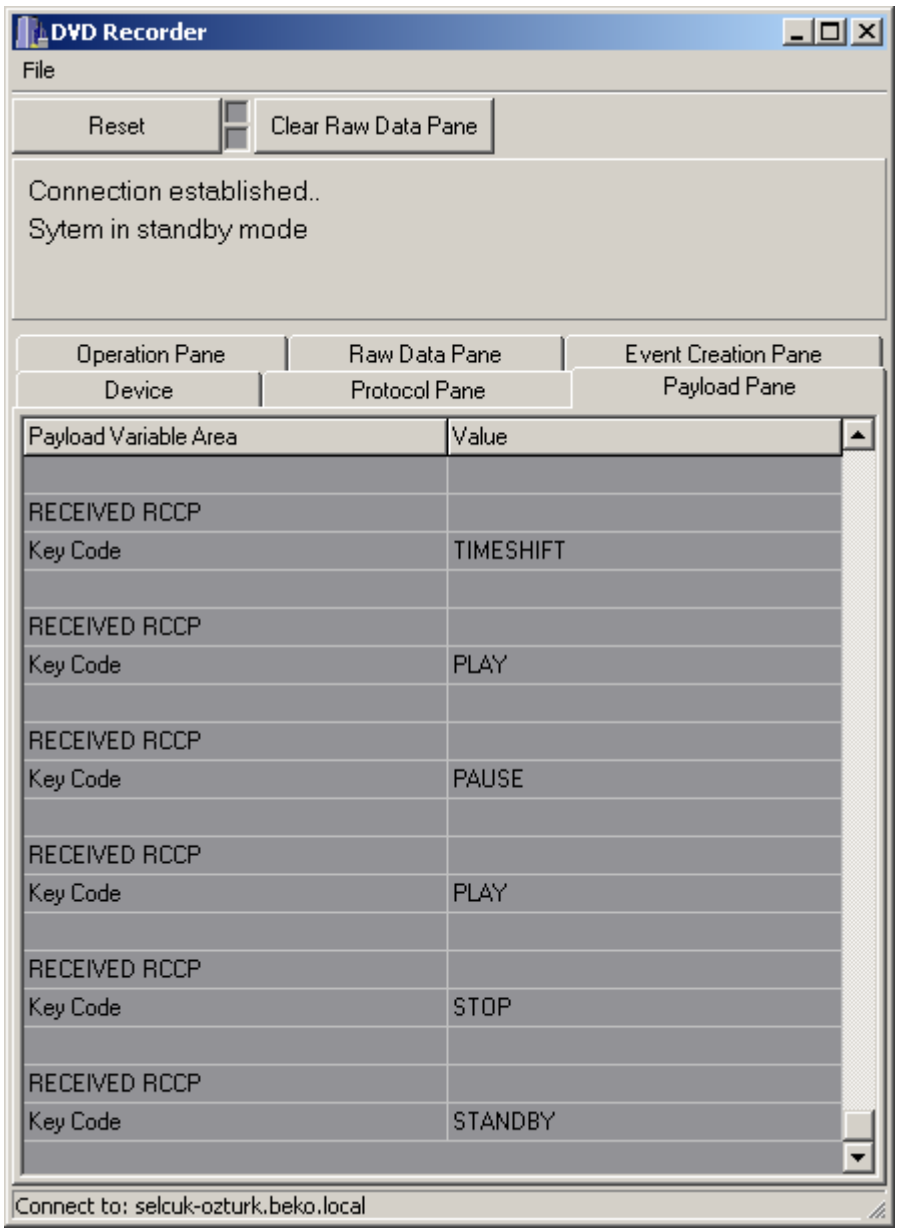


Figure 15 Controllable Device's payload pane after pressing standby key

CHAPTER NINE

CONCLUSION & FUTURE WORK

Remote controllers have started to take an important place in our life recently. The reason for this each device has a remote controller and several features for certain devices can be performed only with the remote controller of that device. Users can control these devices always with the conscious of what thy want to do and which key is pressed then. They may not always have a user manual or a guide to help them.

The increments of the consumer products in houses have brought together many remote controllers with them. As the users have a different remote controller for each device, they need to have each remote controller to be able to control any of these devices. Therefore several remote controller models exist in the market in to be a solution for this problem. However these types of remote controllers can not control the device in the next room.

In this study that has been explained previous chapters, what can be achieved with a new approach for remote controllers and what is required to realize this approach. The most required thing is a communication protocol.

The protocol that called as Smart Adaptive Remote Controller provides an implementation of new approach, completely independent of the communication infrastructure to be used. In order to provide control of more than one device and functioning of these devices based on the principle of changing keys on the remote controller according to the state changes of devices, an infra structure is required that uses Radio Frequency (RF) technique as a communication infrastructure.

The remote controller and the device should face each other while the communication infrastructure infrared. In this study, it is understood that RF communication should be preferred according to the infrared infrastructure, because

of the possibility of state changes of devices at any time due to their features are considered. Furthermore using multicasting method as the principle of our approach makes it compulsory to use a communication infrastructure from radio frequency domain. For example wireless networks are very suitable for this remote controller and its protocol. Because all control mechanisms are implemented and many consumer products are manufactured built-in wireless modules for different applications and these built-in modules can be used for this application or remote controller system.

On the other hand, radio frequency system (for this system wireless module) consumes too much power. While the controllable devices in standby mode these modules may be do not provide the conditions international committee's desired obligations. For that reason a new wireless or radio frequency system must be evolved to provide international committee's desired conditions. Various solutions can be applied for this situation. One of these may be the use of infrared technology to take the devices from stand-by mode. Thus the devices are prevented to consume so much power on stand-by mode. When the device functions in full power mode, new control mechanism can be run with RF technology.

This study now only defines requirements to obtain bidirectional communications and states to achieve adaptive remote controller system. For these a new protocol must be implemented and some messaging structures must be defined. And to provide universality, some user interface libraries must be defined strictly at remote controller side. At the future work, these user interface libraries or widgets should be designed at remote controller side. Also the communication medium and new mechanisms should be defined instead of using wireless networks.

REFERENCES

Cantu, M. (2003). *Mastering Delphi 7.* (1st Edition) Sybex

Celadon Inc. (n.d.) *Sample IR Code Formats.* Retrieved March 10, 2006, from www.celadon.com/infrared_protocol/infrared_protocols_samples.pdf

Dumbill E., Jepson B., Weeks R. (2004). *Linux Unwired.* (1st Edition) O'Reilly

Hyder, D. (2002). *Infrared Sensing and Data Transmission Fundamentals.* Retrieved March 11, 2006 from http://www.onsemi.com/pub/Collateral/AN1016-D.PDF

Harold E.R. (2004). *Java Network Programming*, (3rd Edition) O'Reilly

Kremin, V. (2003). *RC5 Codec.* Retrieved March 11, 2006 from http://www.enee.umd.edu/class/enee445.S2004/rc5codec.pdf

Kernighan, B.W., Ritchie, D.M. (1988). *C Programming Language* (2nd Edition) Prentice Hall

Oualline, S. (1995). *Practical C++ Programming* (1st Edition) O'Reilly

SAA3008 (1988). *Philips Semiconducters DataSheet*. Retrieved July 20, 2006, from http://www.nxp.com/pip/SAA3008_CNV_3.html

SB-Project. (n.d.). Retrieved May 8, 2006, from http://www.xsall.com/~sbp/

Seerden, P. (2003). *Using the Philips 87LPC76x microcontroller as a remote* control transmitter. Retrieved March 10, 2006 from www.semiconductors.philips.com/acrobat/applicationnotes/AN10210_2.pdf

STM3004LD, ST Microelectronics Datasheet
http://www.chipcatalog.com/Datasheet/B825494F2306E69C0C45E18B7CC19F25.htm

Stroustrup B. (1997). *The C Programming Language* (3rd Edition) Addison-Wesley

Yeh, K.W., Wang, L. (n.d.) *An Intro to the IrDA Standard & System Implementation* Retrieved May 13, 2006 from http://www.actisys.com/article.html

APPENDIX A

ABBREVIATIONS

HDLC (High Level Data Link Control)

A Link-Level protocol used to facilitate reliable point-to-point transmission of a data packet.

Note: A subset of HDLC, known as "LAP-B," is the Layer-two protocol for CCITT Recommendation X.25.

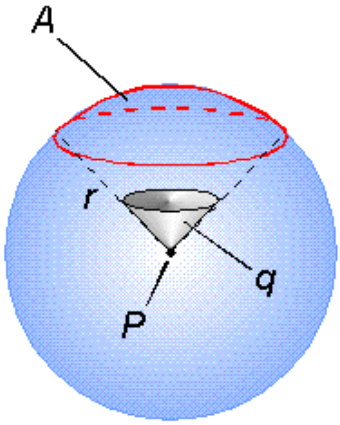
NRZ (Non-Return-to-Zero)

A code in which "1s" are represented by one significant condition and "0s" are represented by another, with no neutral or rest condition, such as a zero amplitude in amplitude modulation (AM), zero phase shift in phase-shift keying (PSK), or mid-frequency in frequency-shift keying (FSK). (188) Note 1: Contrast with Manchester code, return-to-zero. Note 2: For a given data signaling rate, i.e., bit rate, the NRZ code requires only one-half the bandwidth required by the Manchester code.

SR (Steradian)

The metric unit of solid angle. The steradian is the Standard International (SI) unit of solid angular measure. There are 4Π, or approximately 12.5664, steradians in a complete sphere. The name is partly derived from the Greek stereos for "solid".

A steradian is defined as conical in shape, as shown in the illustration. Point *P* represents the center of the sphere. The solid (conical) angle *q*, representing one steradian, is such that the area *A* of the subtended portion of the sphere is equal to r^2, where r is the radius of the sphere.



A general sense of the steradian can be envisioned by considering a sphere whose radius is one meter (r = 1m). Imagine a cone with its apex *P* at the center of the sphere, and that intersects the surface in a circle (shown as a red ellipse, the upper half of which is dashed). Suppose the flare angle q of the cone is such that the area *A* of the spherical segment within the circle is equal to one meter squared ($A = 1m^2$). Then the flare angle of the cone is equal to 1 steradian (q = 1sr). The total surface area of the sphere is, in this case, 12.5664 square meters (4Π times the square of the radius).

Based on the foregoing example, the geometry of which is independent of scale, it can be said that a solid angle of 1sr encompasses about 1/12.5664, or 7.9577 percent, of the space surrounding a point.

The number of steradians in a given solid angle can be determined by dividing the area on the surface of a sphere lying within the intersection of that solid angle with the surface of the sphere (when the focus of the solid angle is located at the center of the sphere) by the square of the radius of the sphere.

The steradian was formerly an SI supplementary unit, but this category was abolished from the SI in 1995.