X input Method for Dzongkha Script

Pema Geyleg
Department of Information Technology
pema.geyleg@gmail.com

Abstract

This paper documents the research conducted while creating Dzongkha keyboard file in XKB to have Dzongkha keyboard support in Linux operating system. Dzongkha Keyboard support means that the user should be able to use the normal English keyboard for inputting Unicode Dzongkha text. This means that an alternate keyboard layout be provided for inputting Dzongkha text. The user should be able to switch/toggle between different keyboard layouts by pressing some combination of keys like ALT+CTRL keys.

It is to be mentioned that it was not possible to cover all the Dzongkha characters in a single keyboard layout as with English characters which have only two levels namely Normal state and Shift state. To overcome this, the Dzongkha keyboard has four levels where by each level displays different characters with each key press. The different four levels are, Normal state, Shift state, Right Alt state and Right Alt + Right Shift state.

XKB is the X server's keyboard module. XKB is a highly configurable module and its behavior is determined by a database of configuration files. These configuration files are read and compiled by the XKB module each time the X server starts. By modifying these configuration files, or adding new configuration files, we can effectively modify the X keyboard behavior to suit our requirements.

1. Introduction

The mechanism for keyboard input under the X window system is described below:

The X server generates events when a key is pressed and when a key is released ie, a Keypress event and a Keyrelease event. The Keyboard gets attached under X to the window or sub-window that has the focus. The X keyboard has two model namely, the server specific codes(called keycodes) and the server independent Symbols(called keysyms).The keycodes represent the physical keys that has to be mapped to ASCII characters before the can be used and the keysyms represents the letters or words that appear on the keys.

For a specific physical key the X server generates a particular keycode. The keycodes for each key are unique. For common alphanumeric keys the keycode generated maybe the same for many workstations but it is not guaranteed to be so. Therefore instead of using the raw keycode the application uses the meaningful characters that are translated by a two step process:

- Firstly the keycodes are translated to symbolic names called keysyms.

- In the second step the keysyms are converted to an ASCII text string which is used to display and save in files and buffers.

The translation of keycodes to keysyms is managed by the X server. Users maybe are able to alter the mapping of one or more keys with utilities such as the "xmodmap". A secondary mapping in the form of keymap tables may also be used to convert the keycodes to keysyms by the client at the application level.

Keyboard mapping in Linux is handled differently for the two modes in which the system can run:

1. Terminal mode.
2. X window system mode.

In terminal mode, the keyboard mapping (i.e. conversion from scan-codes to character codes) is handled by the Linux keyboard driver, whereas in X modes, it is handled by the X server. Both these mappings are implemented by maintaining a mapping table. There are utilities available which allow a user to write his own mapping table (in a particular syntax), these mappings can be then loaded dynamically into the system. This is achieved using X keyboard Extension (XKB) and using Xmodmap mechanism.

Four Levels of Dzongkha keyboard layout is shown diagrammatically below:

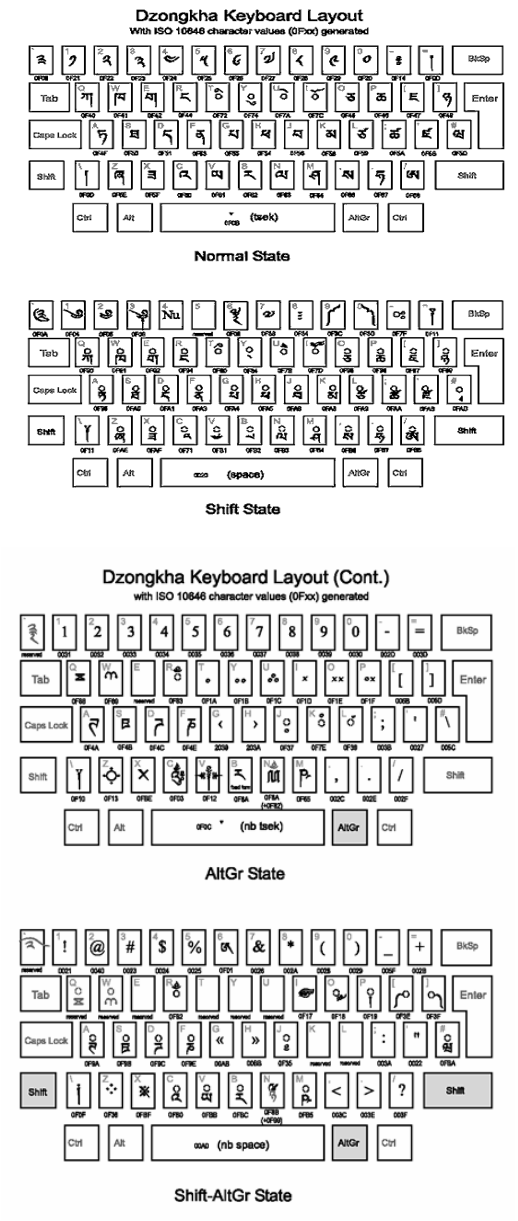

Figure 1: Dzongkha Keyboard Layout

These four levels are listed below:

1. SHIFT State
2. Right ALT State
3. SHIFT + Right ALT State
4. Normal State

This Dzongkha keyboard layout has been prepared at DDA (Dzongkha development authority) while implementing Dzongkha computing capabilities in Microsoft Windows operating system.

However the information on creating the XKB files is only available through the internet. The Xmodmap lets you gain complete control of the keyboard. It is a little hackers' solution. You can map the keyboard to almost anything but this method is discoursged nowadays. The only alternative was to create a dzongkha XKB file.

2. Methods

2.1. XKB Configuration Files

It is a tedious to work through the XKB configuration files. The configuration files are already present, prepackaged and adequate. In the subdirectories keycodes, types, compact, symbols and geometry corresponding to the components of the same name, the main files are present. There are ways to group the main components together into neat bundles. These are held inside the keymap, rules and semantics subdirectories.

2.2. The Basics

Following the basic idea that you can adopt a mix-and-match approach to building a keyboard configuration, the XKB configuration has been decomposed into a number of components. There are sophisticated inclusion and augmentation rules that help to build a component from a basivc set-up and a number of modifications for odd keyboard layouts and national peculiarities. These components are:

2.2.1. Key codes

The keycodes does the translation of scan codes from the keyboard into a suitable symbolic form. At the very bottom of the XKB food chain lays the key codes. The X system generates a key press event and a key release event. This event generates raw numeric keycodes to indicate that a key has been pressed or released.

In both these events, the keycode generated indicates whether a key has been pressed or released. These keycodes components of the XKB assign symbolic names to the various keycodes.

The symbolic names are then used to look up similar keyboard layouts in the symbols component. Below is given something of the structure of a basic keycodes map:

```
xkb_keycodes "basic" {
minimum= 8;
maximum= 255;
<TLDE> = 49;
<AE01> = 10;
<AE02> = 11;
...
indicator 1 = "Caps Lock";
indicator 2 = "Num Lock";
...
alias <AE00> = <TLDE>;
};
```

The “basic” XKB keycodes gives the component type (keycode map) and the variant name (basic). The maximum and the minimum lines after that indicate the maximum and the minimum keycodes generated by the keyboard. It is not a problem if not all the keycodes are used.

Then there are lines like <AE01> = 49; which indicates that keybord keycode (49 in this case) is mapped unto a symbolic keyname (<AE01> in this case). The line here associated a keycode with name that will be used in components such as the symbols component.

The convention that is being used here explicitly names escape-like and shift keys, but names ordinary keycodes by positional code. Therefore, <AE01> is infact 1/! Key on an ordinary QWERTY keyboard.

It is shown in detail below:

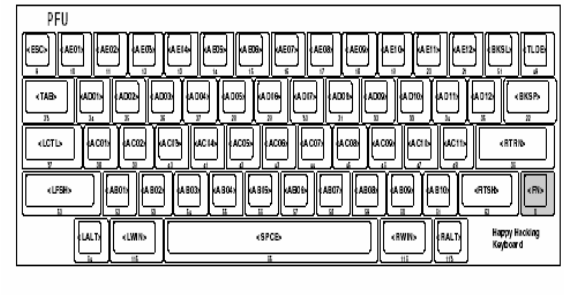

Figure 2: PFU-QWERTY

The left shift key is however denoted by <LFSH>. This convention is used so that the Dvorak-like keyboards can be sensibly specified. The keyboard codes are based notationally on a conventional QWERTY keyboard.

Alternate names can be associated with the same key. This is specified by lines like <AE00> = <TLDE>; shown above. The tilde key associated with keycode 49 in the example above can also be reffered to as <AE00> and also as <TLDE>. This is useful as sometimes you would want to refer to “the right most key of the top row” rather than “the tilde key”.

In the above example, lines like indicator 1 = “caps Lock”; enumerate the indicator LEDs that the keyboard has. From what i think, the name of the indicator is important rather than the number, as it is reffered to in other components like the compact component. It is possible that the numbering of the indicators in the keyboards hardware is reffered to by the indices. This interpretation has the problem that it violates the symbolic = concrete formulation used in the rest of the file.

2.2.2. Key symbols

Pressing a key would produce the actual characters or glyphs ie, the Key Symbols. Symbolic keycodes are mapped unto appropriate symbol by a Symbols map.

Every symbol has a name which is defined as part of the X protocol with a few extra added by the XKB. As far as i can see, the actual symbols name are taken from the /usr/X11R6/includes/keysymdefh file.

The names are listed in this list with the initial XK_ stripped off. For example the name of a simple symbol such as the 'a' is 'a'. Complex symbols such as the B, has names spelt out as the ssharp. A number of odd key symbols are also spelt out according to the XKB protocol specification. Groups and levels can alter the meaning of keys on the keyboard.Therefore, the Symbol map is similar to a matrix which lists multiple symblos for each key. Also the groups and level are used to look up the appropirate symbol.

Below is given something of a symbol map file:

```
partial alphanumeric_keys
xkb_symbols "basic" {
name[Group1]= "US/ASCII";
key <ESC> { [ Escape ] };
...
key <TLDE> { [ quoteleft, asciitilde ] };
key <AE01> { [ 1, exclam ] };
...
modifier_map Shift { Shift_L, Shift_R };
...
};
```

The first line, partial of default alphanumeric keys xkb symbols “basic” indicates that this is a symbol map called basic, along with a few options. The partial option is there to indicate that the map doesnot cover

complete keyboard, only some intresting section of it. The alphanumeric keys option indicates the section of the keyboard that is being covered. The Multiple * keys options are allowed and if any of them has not been specified, it is assumed that the map covers a complete keyboard.

The line name [Group1] = “US/ASCII”; allocates a name to one of the keyboard groups. Other groups can also be specified like wise but a different group name has to be given.

The statement key <TLDE> {[quoteleft, asciitilde]}; indicates a single mapping to a group of symbols (“and” in this case) from a key code (<TLDE> in this case).

If a pair of symbols is enclosed in brackets, it shows that the pair of symbols is seperated by a shift level. Pressing the Shift key would then shift between the two levels, but the types components can overide this. If instead a single symbol is enclosed within brackets, then this symbol is always used independent of the shift level.

The keycode {[symbol,symbol]} is short form syntax of a more general one that allows for a more flexible specification. It is also possible to use inside braces the syntax group [groupname] = [symbol,symbol] instead, with different statements being seperated by commas. This syntax is needed when extra information is being specified for the keys such as the type information.

2.2.3. Compatibility map

The “compatibility map” is usually called the compact component in short. It seems a rather odd name for a component which is mostly concerned with translation of certain key combinations into actions, rather than symbols.

The compact component intercepts certain combinations of keys, usually modifier keys of various sorts. These keys are converted into different actions that range from changing the internal state of the XKB (for example, selecting current group) to moving the mouse pointer. These key strokes are usually consumed by the compatibility map and disappear, even though they can be passed on to other components or to the outside world.

The Compatibility maps also controls the different indicator lights that are displayed. Many Compatibility maps have handling of the modifier keys as a major part. These keys have to be translated into concrete actions-- group and level shifts, etc-- and also locked or latched. Only when it is being held down, a normal (unlocked, unlatched) modifier key has an effect.

A locked modifier key where as, is on until other keypress releases it. Until another key is pressed, a latched modifier key is kept on, at which point the modifier is released. The table below lists various actions in the compatibility map.

Action	Description
NoAction	Do nothing.
SetMods	Set modifier state, while the keys are held down.
LatchMods	Latch modifier state, until the next key is pressed.
LockMods	Lock modifier state, until the keys are pressed again.
SetGroup	Set current group, while the keys are held down.
LatchGroup	Latch current group, until the next key is pressed.
LockGroup	Lock current group, until the keys are pressed again.
MovePtr	Move the mouse pointer.
PointerButton	Also PtrBtn. Simulate a mouse button press.
LockPointerButton	Also LockPtrBtn and LockPtrButton. Simulate a mouse button press, locked until this key is pressed again.
SetPointerDefault	Also SetPtrDflt. Set the default select button?
ISOLock	Convert ordinary modifier key actions into lock actions while this action is active.
TerminateServer	Also Terminate. Shut down the X server.
SwitchScreen	Switch virtual X screen.
SetControls	Set the standard controls, such as slow keys or audible bell. See section 4 of the XKB protocol.[3]
LockControls	Lock the standatd controls.
MessageAction	Also ActionMessage and Message. Generate an arbitrary special-purpose XKB event.
RedirectKey	Also Redirect. Emulate the pressing of a key with a different scan code.
DeviceButton	Also DeviceBtn, DevButton and DevBtn. Emulate an event from an arbitrary input device such as a joystick.
LockDeviceButton	Also LockDeviceBtn, LockDevButton and LockDevBtn. Emulate an event from an arbitrary input device such as a joystick.
DeviceValuator	Also DeviceVal, DevValuator and DevVal. Not implemented.
Private	Generate an arbitrary event with a type and data.

Figure 3: Actions in Compatibility Map

2.2.4. Types

Types give information regarding the levels available for various keys and describe how to shift between the levels.

Each key has its own type and has different number of levels and different ways of switching between these levels; For example when Caps Lock key is on, only alphabetic characters are shifted.

The type a key has can be either being given explicitly in symbols component or is assigned automatically. The follwing rules apply if a type has not been explicitly specified.

- If only one symbol is listed for the key, the type then is ONE LEVEL and there are no level changes.

- The type is ALPHABETC if level 1 symbol is a lower-case letter and level 2 symbols is an upper-case letter. In this case the levels are changed by either the Shift or Caps Lock key.
- The type is KEYPAD if there is a keypad symbol on any level. Here either the Shift or Num Lock key changes levels.
- Otherwise, it is of type TWO LEVEL and the Shift key changes the levels.

Following gives an example type's component:

```
partial default xkb_types "default" {
virtual_modifiers LevelThree;

type "THREE_LEVEL" {
modifiers = Shift+LevelThree;
map[None] = Level1;
map[Shift] = Level2;
map[LevelThree] = Level3;
map[Shift+LevelThree] = Level3;
level_name[Level1] = "Base";
level_name[Level2] = "Shift";
level_name[Level3] = "Level3";
};
};
```

The line virtual modifiers LevelThree; indicates that the virtual modifiers are to be used. These are modifiers that have an equivalent virtualMods = LevelThree somewhere in the symbols component or somewhere in the compact component. The standard shift modifiers need not be specified.

The line modifiers = Shift + LevelThree; assigns the set of modifiers that are to be considered for this particular type.

The line map[Shift] = Level2; describes that level 2 is to be used whenever shift key is down.

The level name[level2] = “Shift”; line are those that assigns suitable names to each level. These are useful when you want to do things like printing out key maps. Besides that they do not have any direct effect on any thing.

2.2.5. Geometry

Geometry describes the physical layout of a keyboard. There exist a number of other components such as rules, semantics, keymaps that are essentially ways that packages the main components into more usable collections.

However, the geometry files are some of the most complex for something that is so trivial. Below is given a sample file:

```
default xkb_geometry "pc101" {
description= "Generic 101";
width= 470;
height= 210;
...
shape.cornerRadius= 1;
...
shape "NORM" { { [ 18,18] }, { [2,1], [ 16,16] } };
...
solid "LedPanel" {
shape= "LEDS";
top= 52;
left= 377;
color= "grey10";
};
...
indicator "NumLock" { left= 382; };
...
```

The line default xkb geometry “pc101” has the usual options ie, type and name declaration. The line description = “Generic 101”; assigns a descriptive name to the keyboard.

The total width of the keyboard is given by the line width=470;. All of the lengths in geometry declarations are multiples of 1mm, therefore 470 is 47cm. The shape.cornerRadius = 1; line indicates default setting for a shape field. Here in this case, the corner radius of a join is set to 1mm. Listed below are other items such as solids that have other fields.

The line shape “NORM” {{[18,18]}, {[12,1]}, {[16,16]}}; declares a shape. A shape can be used to draw something like a key or indicator. It is a named drawing outline.

This shape above defines the look of a normal key. All of the coordinates are starting from the upper left corner and increases rightwards and downwards. The position of the shape is shifted wherever the shape needs to be drawn from the origin of the shape which i always (0,0). A Shape consists of list of outlines and each one represents a closed figure. By outline, it means a list of coordinates in [x,y] form with each list enclosed in braces.

An outline interpretation depends on the number of coordinates in the list. If only one coordinate is given, the box is drawn from origin (0,0) to that coordinate. If two are given in the list, then the box is drawn from the first to the second coordinate. If three or more are given in the list, then an arbitary closed figure is drawn

with vertex at each coordinate. In the above example, a box is drawn from (0,0) to (18,18). Another from (2,1) to (16,16) to give a suitable looking key outline.

From above, the section beginning with solid "Ledpanel" draws a solid area of color. A solid serves as an example of a doodad; which is a piece of decoration for filling out the appearance of the keyboard. Other example of doodads are indicators, outlines, text and logos. The fields of the solid are left and top, shape and color. lLeft and top gives the start position of the solid. Shape gives a named shape for the solid to draw. Color gives the color of the solid. which can be any of the named X11 colors.The field priorty assigns the drawing order for overlaid items.

The line indicator "NumLock" {left= 382 ;}; declares a suitably named doodad. For an indicator the possible fields are: onColor, offColor, left, top, priority and shape. I am not sure entirely about how an indicator gets mapped onto an appropirate logical indicator from the compact or the keycode components. It could be related to the numbering of the indicators.

2.2.6. Modifier keys

The modifier keys are those keys that are used to change the meaning of other keys. Examples include the Shift, Control or Alt key. The can also be combined to give combinations like the Control+Shift+Alt. This handling of extended modifier key combinations is what makes the XKB, to a certain extend so complex.

XKB recognizes eight modifier keys at the base level namely, Control, Shift and Lock keys and the generic Mod1-Mod5 keys. These keys correspond to the keys in the core X protocol. It needs to be there so that the older programs can understand whats going on. Keys such as the ubiquitous Alt keys are mapped onto one of the Mod keys. Its all well to have the basic modifier keys but it would also be handy to be able to introduce a level of abstraction. In this way you would be able to talk about the modifier keys by function rather than by explicit keyname. With XKB virtual modifier keys can also be used where a basic modifier key or combinations is mapped onto a named virtual modifier. The behavior of the keyboard is described by the virtual modifiers, decoupling the exact physical capabilities of the keyboard you are using from the sort of characters that you want to type. This side of things are largely handeled by the compact components.

An example is shown below.

```
key <RALT> {
symbols[Group1]=
[Mode_switch, Multi_key ],
virtualMods= AltGr
};
```

2.2.7. Levels and Groups

A level stands for the sort of things that a Shift key is normally expected to do. Normally, you would expect an 'a' character to appear when the key marked 'A' is pressed and 'A' character to appear instead when you hold down the Shift key and press the same key. Normally the two levels, Shift and non Shifted are enough. The fairly standard way of shifting between levels is to press the Shift or the Lock keys.This nice state of affairs is complicated by keys such as the Return key and keys that often have more complex levels such as numeric keypad.

Groups are more slippery concept in contrast to levels that are staraightforward. With groups you can shift the entire keyboard over to some other character set. This way you would be able to access characters which are not usually considered part of the standard keyboard.

To shift groups the keys needed are less obvious than the keys needed shift levels because there isn't any immediate equivalent to the shift key and the Alt key is assigned other duties. It is upto the user to choose from the number of possible combinations that the standard XKB configuration files. There are multiple levels inside each groups with each level reflecting the suitable shifts for the characters in the groups.

There are also upto 64 Shift-Levels for each group. Normally two are used: level for un-shifted keys and the second one: shifted keys. In Dzongkha keyboard layout up to four levels are used.

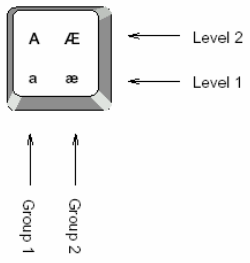

Figure 4: Levels and Groups on a single key

2.2.8. Handling groups

By including other list of symbols after the first list, Multiple groups can be specified. Each of these list are seperated by commas. Each group has to be named and below is shown what the resulting configuration should look like.

```
name[Group1]=”US/ASCII”;
name[Group2]=”Dzongkha”;
…….
Key <AD01> {[q, Q], [U0F40, U0F92]};
```

Two groups are defined in this case. Here for the Q key, two lists of symbols are given. One list is for the US/ASCII group and the other for the Dzongkha group. There are also possibilities to specify multiple groups using the long form syntax as shown below.

```
Key <AE11> {
symbols[Group1]=[ minus , underscore ],
symbols[Group2]=[ minus , questiondown ]
};
```

It is also possible that a partial symbol map that only provides mappings for some higher, non default group may be defined. The symbol mappings for the lower level groups are empty in this case. These sort of symbol maps can also be included with more basic maps. For example the line below defines a group 3 (only) Mapping.

```
Key <AD01> {[],[],[q,Q]};
```

2.2.9. Handling Levels

It is the types component that specifies how differing levels are to be handeled for various keys. It would appear that by default, most keys are automatically given a two level or alphabetic type. However, in some cases, it may be necessary to explicitly associate a level behavior with a key. Here the type and the symbol map needs to explicitly given. Below is an example:

```
key <PRSC> {
type= "PC_SYSRQ",
symbols[Group1]= [ Print, Sys_Req ]
};
```

From the above example, it is shown that the <PRSC> keycode is given an explicit type (PC SYSRQ). The group 1 symbols then are given explicitly using the long syntax.

2.2.10. Control Keys

Generally, a low value control key character is expected to be produced by holding the Control key down and typing and alphabetic key. For example, ASCII 8 is produced by Control H. The information about which keys produce which control characters are hard wired and specified in the XKB protocol. Which are the control keys are specified by using the modifier map statement to map keys onto the Control modifier. Below is given an example.

```
modifier_map Control { Control_L };
```

2.2.11. Special Characters

There are huge array of “symbols” that are intended to be used instead to control various parts of the XKB or the X11. The modifier keys discussed above are the most obivious. I addition to that there exists a range of symbols designed to do things like launch a web browser or be combined with other symbols. The character codes are present for all of these symbols.

2.2.12. Dead keys

The dead keys are keys that are intended to represent accents which are combined with other symbols to form an accented symbol. For example, the keyname’ key and the a key can be combined to form the á symbol. Nothing immediately appears when a dead key is pressed. An XKB will instead wait for the next character and will attempt to combine it with the accent. Names like the dead_acute are given to dead keys.

2.2.13. ISO keys

There are a grab-bag of features provided by the ISO keys: keys for shifting group and level, for setting word processor like features (such as margins) and for moving about the screen. An ISO prefix is given to all of these keys.

X11 Control keys

Here the control keys to terminate the X server and keys to flip between virtual screens are included.

2.2.14. XKB Control keys

For ease of use, the XKB provides a number of facilities: options to handle slow typing,
key repeat, sticky keys and using keys to move the mouse pointer. There are also pseudo-keys such as **Mode_shift** that are not intended to actually produce symbols in addition.

Give the research ***methodology*** there. Methods present all the details of the study methodology, e.g. the subject, research design, statistics etc.

These are the keys that the compact component consumes to produce actions.

2.2.15. Composition keys

The accented characters and the like can be build by using the Multi_key key. It is produced by pressing the Multi_key key, followed by the letter, followed by the accent character.

The list of compose sequence for a particular locale or encoding (iso8859-2 for example) and what the produce is contained within the /usr/X11R6/lib/X11/locale/localename/Compose.

2.3. XKB Programs

A number of programs are there which come with XKB for management and debugging purposes. All of these have adequate man pages and have a sensible –help option. I'll just be metioning a few things that may be useful here.

2.3.1. Setxkbmap

The Setxkb map program allows an XKB map to be installed. The Components can be directly specified through arguments such as –symbols en US(pc104)+Dz or by using the rules parameter.

2.3.2. Xkbcomp

The Xkbcomp compile an XKB keyboard description. This is the program that converts the contents of a suitably specified set of configuration files into a suitable form for the X-server to use.
The intresting thing about this program is that it can also be used to produce a source file for the current XKB configuration, using the –xkb configuration. To give an example, command xkbcomp: 0.0 -xkb will produce a file called server-0 0.xkb that contains the complete configuration source for server 0.0.

2.3.3. Xkbprint

The Xkbprint produces a graphical map of the keyboard. The map shows the keys for each character and is usually a Postscript map of the keyboard. For example a command such as xkbprint:0.0 will produce a file called server-0.0.ps with the map of the basic keyboard. To allow things such as the scan codes or key names to be printed, options such as the –label type is used instead.

3. Results

3.1. Dzongkha XKB file named dz

```
partial default alphanumeric_keys
xkb_symbols "basic" {
      name[Group1]= "Dzongkha"; //consists of
1 group only

      key <TLDE> { [      0x1000F09,
0x1000F0A, 0x1000F6C, 0x1000F6D ]     };

// numbers e.a.
      key <AE01> { [      0x1000F21,
0x1000F04, 1, exclam ]       };
      key <AE02> { [      0x1000F22,
0x1000F05, 2, at     ]       };
      key <AE03> { [      0x1000F23,
0x1000F06, 3, numbersign ]   };
      key <AE04> { [      0x1000F24,
0x1000F48 , 4, dollar  ]     };
      key <AE05> { [      0x1000F25,
0x1000F70 , 5, percent ]     };
      key <AE06> { [      0x1000F26,
0x1000F08, 6, 0x1000F01 ]    };
      key <AE07> { [      0x1000F27,
0x1000F38, 7, ampersand ]    };
      key <AE08> { [      0x1000F28,
0x1000F34, 8, asterisk ]     };
      key <AE09> { [      0x1000F29,
0x1000F3C, 9, parenleft ]    };
      key <AE10> { [      0x1000F20,
0x1000F3D, 0, parenright ]   };
      key <AE11> { [      0x1000F14,
0x1000F7F, minus, underscore ]        };
      key <AE12> { [      0x1000F0D,
0x1000F11, equal, plus ]     };

// consonants ( ka kha ga nga )
      key <AD01> { [      0x1000F40,
0x1000F90, 0x1000F88, reserved  ]     };
      key <AD02> { [      0x1000F41,
0x1000F91, 0x1000F89, reserved  ]     };
      key <AD03> { [      0x1000F42,
0x1000F92, 0x1000F6E , 0x1000F6F]     };
      key <AD04> { [      0x1000F44,
0x1000F94, 0x1000F83, 0x1000F82  ]    };

// vowels ( i u e o )
      key <AD05> { [      0x1000F72,
0x1000F80, 0x1000F1A, reserved  ]     };
      key <AD06> { [      0x1000F74,
0x1000F84, 0x1000F1B, reserved  ]     };
```

```
      key <AD07> { [      0x1000F7A,
0x1000F7B, 0x1000F1C, reserved ]      };
      key <AD08> { [      0x1000F7C,
0x1000F7D, 0x1000F1D, 0x1000F17  ]    };

// consonants ( ca cha ja nya )
      key <AD09> { [      0x1000F45,
0x1000F95, 0x1000F1E, 0x1000F18   ]   };
      key <AD10> { [      0x1000F46,
0x1000F96, 0x1000F1F, 0x1000F19   ]   };
      key <AD11> { [      0x1000F47,
0x1000F97, 0x100005B, 0x1000F3E ]     };
      key <AD12> { [      0x1000F49,
0x1000F99, 0x100005D, 0x1000F3F  ]    };

// consonants ( ta tha da na pa pha ba ma tsa
tsha dza wa )
      key <AC01> { [      0x1000F4F,
0x1000F9F, 0x1000F4A, 0x1000F9A ]     };
      key <AC02> { [      0x1000F50,
0x1000FA0, 0x1000F4B, 0x1000F9B ]     };
      key <AC03> { [      0x1000F51,
0x1000FA1, 0x1000F4C, 0x1000F9C ]     };
      key <AC04> { [      0x1000F53,
0x1000FA3, 0x1000F4E, 0x1000F9E ]     };
      key <AC05> { [      0x1000F54,
0x1000FA4, 0x1002039, 0x10000AB ]     };
      key <AC06> { [      0x1000F55,
0x1000FA5, 0x100203A, 0x10000BB ]     };
      key <AC07> { [      0x1000F56,
0x1000FA6, 0x1000F37, 0x1000F35 ]     };
      key <AC08> { [      0x1000F58,
0x1000FA8, 0x1000F7E, reserved ]      };
      key <AC09> { [      0x1000F59,
0x1000FA9, 0x1000F39, reserved ]      };
      key <AC10> { [      0x1000F5A,
0x1000FAA, 0x100003B, 0x100003A ]     };
      key <AC11> { [      0x1000F5B,
0x1000FAB, apostrophe, quotedbl ]     };

// TODO: BKSL and LSGT are from gb layout,
are there any variants?
      key <BKSL> { [      0x1000F5D,
0x1000FAD, 0x100005C, 0x1000FBA ]     };
      key <LSGT> { [      0x1000F0D,
0x1000F11, 0x1000F10, 0x1000F0F ]     };

// consonants ( zha za 'a ya ra la sha sa ha
a )
      key <AB01> { [      0x1000F5E,
0x1000FAE, 0x1000F13, 0x1000F36 ]     };
      key <AB02> { [      0x1000F5F,
0x1000FAF, 0x1000FBE, 0x1000FBF ]     };
      key <AB03> { [      0x1000F60,
0x1000F71, 0x1000F03, 0x1000FB0 ]     };
      key <AB04> { [      0x1000F61,
0x1000FB1, 0x1000F12, 0x1000FBB ]     };
      key <AB05> { [      0x1000F62,
0x1000FB2, 0x1000F6A, 0x1000FBC ]     };
      key <AB06> { [      0x1000F63,
0x1000FB3, 0x1000F8A, 0x1000F8B ]     };
      key <AB07> { [      0x1000F64,
0x1000FB4, 0x1000F65, 0x1000FB5 ]     };
      key <AB08> { [      0x1000F66,
0x1000FB6, comma, less ]     };
      key <AB09> { [      0x1000F67,
0x1000FB7, period, greater ] };
      key <AB10> { [      0x1000F68,
0x1000FB8, slash, question ] };

// space
      key <SPCE> { [ 0x1000F0B, space,
0x1000F0C, 0x10000A0 ] };

//to enable Right Alt key to access the 3rd
level
     include "level3(ralt_switch_multikey)"
};
```

Character set used in Dzongkha script is the same as that has been assigned for Tibetan script within the U0F00 – U0FCF Unicode range. It is depicted diagrammatically in Appendix A.

Results section presents the results of the study in a form which conveys their meaning. Consider the best way of presentation. You may use figures: figures should be self-explanatory with the corresponding caption; units, axes, and legends. You may also use tables: tables should be self-explanatory with the corresponding caption; units, Word descriptions. If the data results are large, only put a summary of the significant research finding in this section.

4. Discussion

The Dzongkha XKB file was successfully tested and is being used in the Linux operating system. With the successful installation of Dzongkha keyboard and Dzongkha fonts in the Linux operating system, one will be able to type Dzongkha Unicode characters using “gedit” editor. However complex multi stacked characters cannot be rendered properly as we need a renderer for Dzongkha script. That is we need to have Pango module with Dzongkha script support to have proper rendering in Gnome application and ICU layout engine defined for Dzongkha script to properly render Dzongkha characters in Open Office application.

We are in the later stage of development concerning the creation of Pango module support for Dzongkha script. It can be said that we have successfully developed a Pango support for Dzongkha script which is being tested in terms of its reliability with other applications on Linux operating system. Studies are being conducted on developing Dzongkha Layout Engine for ICU along with the work of localizing Open office 1.1.1.

5. Conclusion

X input method for any language can be successfully created using X keyboard extension (XKB) and Xmodmap with the exception that their script is supported in Unicode. However most people discourage the use of Xmodmap method for creating keyboard support for one's language. Then it is to be noted that with the successful installation of fonts and

keyboard driver doesn't guarantee proper rendering of the complex scripts as explained above.

Then there is another form of creating input method called IIIMF (Intranet Internet Input method) and it is out of scope of this research. This method has been carefully studied and it is to be implemented at Sherubtse College as a senior project for Computer students.

6. References

[1] T Karoonboonyanan. "Thai Input Method Implementations", http://linux.thai.net/thep/th-xim.

[2] I Pascal. "X Keyboard Extension" http://pascal.tsu.ru/en/xkb

[3] D Palmer. "Unreliable Guide to XKB" http://www.charvolant.org/~doug/xkb