Xshell

Demonstration Guide

Table of Contents

Welcome to Xshell

Introduction

Xshell provides a simple shell interface, editor and interpreter designed for the Xmos XGC board using a single-core Xmos XS1-L1 device.

Aims

The aim of this project was to demonstrate how Xmos's unique architecture can be utilised to power or use typical computer components without requiring any dedicated processing chips, hence cutting down the cost of hardware and providing more flexibility due to the “software-defined silicon” approach.

About the XGC Board

The XGC board features a single-core XS1-L1 processor, VGA connector, PS2 keyboard/mouse connector, 1 bit speaker, SDRAM module, SD card reader/writer, 3 push buttons and a directional joystick.

Xshell Features

The software provides the following features:

- Text mode VGA (640x480px) driver with graphics routines.
- Keyboard driver with blocking and non-blocking modes.
- Xmos SD Card Driver (from XmosLinkers) with further text-orientated routines.
- Simple speaker output driver.
- Shell featuring SD card manipulation and interactive interpreter.
- Full interpreter.
- Simple editor allowing saving and loading to SD cards.

Using Xshell

Introduction

This section describes how to use the Xshell demonstration shell.

Types of Commands

In the shell any command started with a '/' character is considered a system command. These commands will be run by the shell and not the interpreter. Any command that is not started with the '/' character will be passed to the interpreter.

System Commands

Below are the valid system commands:

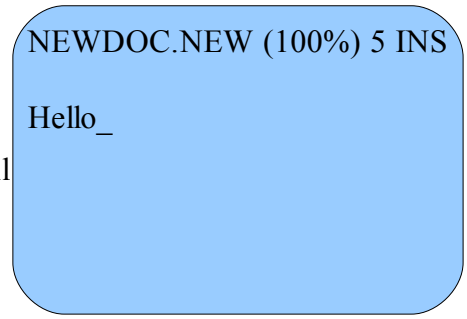
Command	Description
/sdcheck	Check if an SD card is inserted and print its status.
/sdcat	List the files on the inserted SD card.
/sdmore [file]	Print the contents of [file] to the screen.
/sdprint [text]	Print the text [text] to a file called PRINT.TXT on the card.
/sddel [file]	Delete the file [file] from the SD card.
/sdtouch [file]	Check if the file [file] exists.
/cs	Clear the screen.
/editor ([file])?	Start the editor with an optional [file] to edit.
/sdinterpret [file]	Interpret the file [file] from the SD card.
/clearistate	Clear the state of the interactive interpreter.

Using editor

Introduction
This section describes how to use the text editor, editor.

Limitations
Editor can only edit text files up to 3KB in size. Attempting to load a file larger than 3KB will cause an error.

Interface
Editor provides two modes, INS (insert text) and CMD (command). Press the [ESC] key on the keyboard to toggle between them. The top right hand side shows whether the editor is in command or insert mode. When in command mode a '>' will appear at the bottom of the editor allowing you to insert text.



The top line also shows the filename, position in the file (in %) and the total number of characters in the file.

Command mode commands
The following commands are available in command mode:

Command	Description
load [file]	Load [file] into the editor.
save ([file])?	If [file] is specified, save into the file [file]. If [file] is not specified, save into the current file.
quit	Return to the shell, discarding all changes.

If there is a problem with an operation then the editor will warn you with “!” and an error.

Starting editor from Xshell
You can start the editor from Xshell using the /editor command. Note the following:

- /editor starts the editor with no default file open.
- /editor [existingfile] opens [existingfile] for editing.
- /editor [newfile] opens [newfile] for editing and prepares to create the file on disk.

Interpreter

Introduction

This section describes detailed to use Xshell's simple interpreted language to create applications for the XGC board.

Operation

The interpreter takes the Xshell language, preprocesses it into an easy-to-execute intermediate code and then executes it on the processor. The interpreter takes two part instructions of the form COMMAND PARAMETER. There must be *no spaces* in the command or parameter, but unlimited whitespace is allowed in other places.

Limitations

In Xmos XC memory is normally pre-allocated. As such the interpreter places limitations on the size of programs, number of variables, nested loops etc to prevent overflows. These can be modified by editing the header file xgc_interpreter.h:

Pre-processor Directive	Default	Description
`MAX_VAR_NAME_LEN`	10	Maximum length of a variable name.
`MAX_TOKEN_LEN`	50	Maximum length of a single command or parameter.
`NO_OF_VARS`	30	Number of variables available to programs.
`NO_OF_LOOPS`	20	Number of levels of nested loops and conditionals (ifs).
`MAX_TOKENS`	250	The maximum number of *intermediate code* tokens in the program. There is not a direct one-to-one correlation between the number of commands in and the number of intermediate code tokens out.

Using the Interactive Interpreter

Any command(s) typed at the Xshell prompt that do not start with '/' will be treated as a programming language construct. The state of, for example, variables will be kept constant throughout the Xshell session so commands can be executed interactively.

Note: Using the file interpreter will wipe the state of the interactive session. The state of the file interpreter on termination will be inside the interactive interpreter as this is useful for debugging.

Note: You can explicitly reset the state of the interactive interpreter by using the system command /clearistate .

Using the File Interpreter

Any SD card file can be interpreted by the interpreter by using the /sdinterpret [file] command. This will clear the state of the interpreter and interpret the file, returning to the Xshell command line after termination.

Language Expressions

The simple interpreted language can only take certain types of expressions. The table below describes the valid types of expressions.

Expression	Description
Arithmetic Expression arithexp	An arithmetic expression can only feature one or two operands of the format: int | variable | ((int|var)OP(int|var)) Supported operators are: + - * / Examples of valid expressions: 1 hello (1+hello)
Boolean Expression boolexp	A boolean expression is one or more two operand boolean expressions separated by AND or OR. ((int|var|specialid)OP(int|var|specialid) [(and|or)(int|var| specialid)OP(int|var|specialid)]*) Supported operators are: == != < > <= >= Supported AND/OR operators are: & | A Special ID is a reserved keyword that represents a useful thing on the XGC board or an external device. Please see later sections. Examples: (x==1) (y!=2) ((x!=1)&(y!=2))

Note that spaces are not permitted in expressions.

Language Constructs

Below are the available language constructs:

Construct Syntax	Description
beep [time - arithexp]@[freq - arithexp]	Causes the speaker to beep for time * 1/10 seconds at a frequency of freq in Hz. Eg beep 10@1000
sleep [time - arithexp]	Causes the program to sleep for time * 1/10 seconds. Eg sleep 1
set [var]=[arithexp]	Sets variable to an arithmetic expression. Eg set x=10 set x=(x+1)
setcur [row – arithexp]@[cols - arithexp]	Set the graphics cursor to row, column as character-width increments. Eg setcur 10@20
print [variable]|'[char]'	Print the value of a variable to the current cursor position OR print the character to the current cursor position. Eg print x print 'x'
println	Insert a new line into the printed text. Eg println
printnc [arithexp]	Interpret the expression as a ASCII character code and print it to the screen. Eg printnc 32 printnc (x+2) IMPORTANT: Using printnc on a value outside the range of the ASCII character set will cause it to be ignored.
getkey [variable]	Get the most recently pressed key and store it in variable. If no key has been pressed, stores -1. Eg getkey x
random [variable],[max - arithexp]	Stores a “random” number between 0 and max in the specified variable. Eg random x,10
if [boolexp] *[block1]*	If boolexp is true, execute block1. If boolexp is not true, execute block2 if specified or end.

(else *[block2]*)? endif	Eg if (x==1) print x else print y endif
while [boolexp] *[block]* endwhile	If boolexp is true, execute block. Repeat until false. Eg while (x==-1) getkey x endwhile

Example – printing input from the keyboard

Functions available in the language can make using the display and keybord easy. Consider the example below.

```
set x=-1
while (x!=key_enter)
    if (x!=-1) printnc x endif
    getkey x
endwhile
```

This program uses the printnc function, which interprets an integer as a character and the getkey function to get a key from the keyboard. It also uses the key_enter specialid (see later) which refers to the enter key on the keyboard. It is also very solid, as printnc will ignore all values outside the range of the ASCII character set, and as such the behaviour of the program is as expected.

Special IDs

The language provides a set of "Special IDs" which are reserved keywords that can refer to useful physical devices on the XGC.

There are two sets of Special IDs: left hand side and right hand side. LHS refer to physical buttons on the XGC board and must be enabled by declaring the ENABLE_XGCBUTTONS preprocessor directive when compiling Xshell. *This will decrease the performance of the intermediate code.* RHS refers to keyboard keys such as the arrows and enter keys. There is no performance penalty when using RHS Special IDs.

LHS	RHS
BUTTON_A	KEY_ENTER
BUTTON_B	KEY_SPACE
BUTTON_C	KEY_UP
BUTTON_JC	KEY_DOWN
JOY_UP	KEY_LEFT

JOY_DOWN JOY_LEFT JOY_RIGHT	KEY_RIGHT

In the keyboard example above, key_enter is an RHS Special ID.

Note on Graphics

Each character is an 8x8 square, giving 0 to 79 (cols) and 0 to 59 (rows). Each character can take a value from the ASCII character set only. Note that you cannot issue the command print ' ' due to restrictions on the language; use printnc 32 instead.

Included Examples

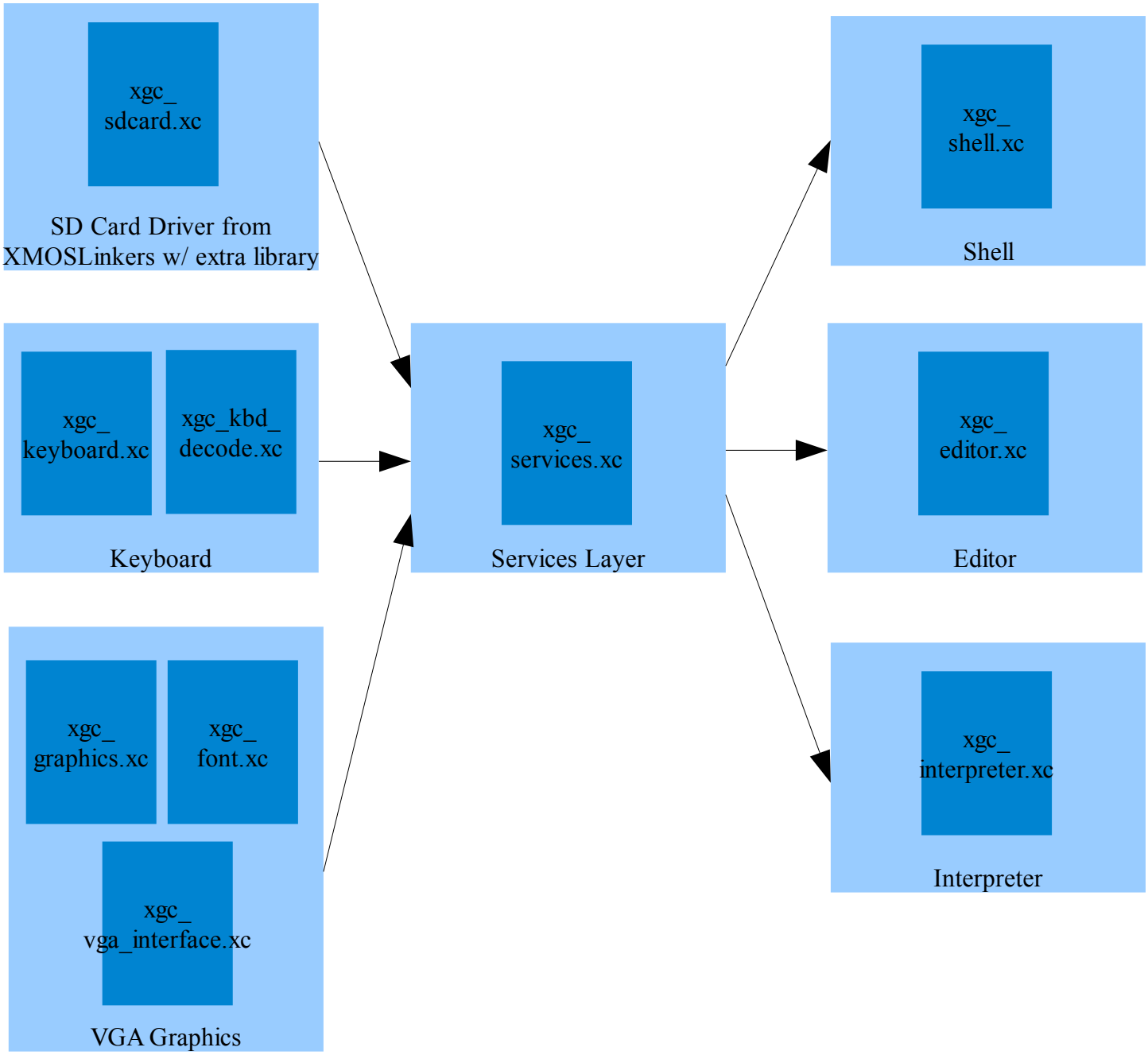
KEYP.TXT	Prints text from the keyboard onto the screen, as shown above.
PATE.TXT	Prints a pattern of random characters onto the screen.
SPLB.TXT	Plays a simple ball game with the user. Press any key to start at the beginning. Note that this example takes a while to load.
MOLE.TXT	Play a game of “hit a mole” using the number pad on the keyboard. When the mole's eyes appear, press the corresponding key on the number pad for the position. Note that this example takes a while to load. Press the enter key to start the game.

Program Organisation

Introduction

This section provides an overview how the program is organised and a description of its components.

Block diagram



The lower-level aspects of the application are accessed through a services layer which abstracts the services they provide. The service layer uses a channel-based “send command, length and then data” protocol, however many operations can be accessed through a set of functions. When using raw the format is approximately channel_to_services <: CONTROL_TOKEN, channel_to_services :> receive_reply.

Service Layer Functions

The tables below summarises the tokens and functions available in the services layer.

Control Token	Description
KA_SYS_WAKEUP AK_SYS_FINISHED AK_SYS_GIVE_CONTROL	To be used for switching processes (no longer implemented).
AK_KEY_SETMODE	Set the mode for keyboard input. 0 is blocking (wait for a key to be pressed and return it) while 1 is nonblocking (return -1 if no key is available). Mode should be set if you use nonblocking.
AK_KEY_REQ	Request a blocking key.
AK_KEY_NBREQ	Request a nonblocking key.
AK_GRAPHICS_ADD_CHAR	Add the following character to the current position on the screen. **services_graphics_add_char** **services_graphics_add_text**
AK_GRAPHICS_NEW_LINE	Insert a new line at the current position on the screen. **services_graphics_new_line**
AK_GRAPHICS_DEL_PREV_CHAR	Delete the previous character before the current position on the screen.
AK_GRAPHICS_CS	Clear the screen. **services_graphics_cs**
AK_GRAPHICS_SET_LINE	Set the current line (rows) **services_graphics_set_line**
AK_GRAPHICS_SET_CHAR	Set the current char (cols). **services_graphics_set_char**
KA_SDCARD_OKAY	Returned by the service to indicate the SDC operation was OK.
KA_SDCARD_ERROR	Returned by the service to indicate the SDC operation was not OK. Followed by an error code.
AK_SDCARD_CAT	Returns an array containing the filenames of all the files on the SDC. **services_sdcard_cat**
AK_SDCARD_READ	Read a file from the SDC: Token->filename size->filename chars **services_sdcard_load**
AK_SDCARD_WRITE	Write to the SDC: Token->filename size->filename chars->Data size->Data chars **services_sdcard_write**
AK_SDCARD_DEL	Delete to the SDC (as per read).

	services_sdcard_del
AK_SDCARD_INSERTED	Return if the SDC is inserted. **services_sdcard_inserted**
AK_SDCARD_EXISTS	Return if a file exists (as per read). **services_sdcard_exists**
AK_SPEAKER_TONE	Output a tone to the speaker, expects token->length in 1/10ths of a second->freq in Hz
AK_GET_INPUT_STATUS	Returns a 32 bit integer with bits corresponding to the status of the buttons and joystick on the XGC.

Acknowledgements

SD Card Driver

By coz and Larry at XMOSLinkers, http://www.xmoslinkers.org/node/227

Keyboard Key Decode Routine

Heavily modified from Roger Shepherd's code, http://www.xmoslinkers.org/node/235/