*Uni*ROM Library Documentation

UNIROM.LIB contains procedures that allow one to write robust custom front-end applications for *Uni*ROM. Our own URLOAD and UREDIT programs were written and linked with this library.

We start off with an overview of general Library usage and follow that with a few examples. The final sections provide complete documentation of each routine in the Library.

General Usage

Use of the *Uni*ROM Library is very straight forward.

All *Uni*ROM applications must perform the following actions:

- Fill in the port structure (manually or from a script through "get_port_cfg()")
- Call ur_init() to initialize the communications port
- Call ur_select() to activate a specific *Uni*ROM

Most applications would then do the following:

-Fill in a CFG structure with setup data (manually or from a script through "get_cfg_section()")
-Call ur_wt_cfg() to setup this *Uni*ROM

If the application's purpose requires fully arbitrated access to *Uni*ROM's dual-ported memory, it would then call ur_reset_target(OFF) to release reset. If the application's purpose is more static in nature (like a file loader) it would call ur_reset_target(ON) to insure the fastest possible accesses.

At this point, the application can make multiple *Uni*ROM library functions to access *Uni*ROM's features.

If the application uses multiple *Uni*ROMs, it can repeat the steps from "Call ur_select()" on for each *Uni*ROM.

The library routines hook CONTROL-C, CONTROL-BREAK and EXIT() to automatically release allocated memory and restore interrupts before the application shuts down.

Examples

The Library sub-directory on your distribution disk contains several examples that use the *Uni*ROM Library. These examples were chosen to demonstrate typical uses of the Library. Each example is fully documented with comments.

The first example (example1.c) is a very general demonstration of how to initialize the Library and perform several function calls to perform typical operations. It uses hard-coded parameters for all configurations.

DUMP.C demonstrates use of the script parsing commands to use port definition information and *Uni*ROM configuration information from a script file. DUMP.C retrieves a block of data from *Uni*ROM and then formats and displays it.

LDUMP.C demonstrates LIVE arbitrated access to *Uni*ROM's emulation memory while the target is executing out of the same memory. LDUMP.C performs the exact same function as DUMP.C except it retrieves the data WHILE THE TARGET IS RUNNING.

WATCH.C demonstrates another application for fully arbitrated access to *Uni*ROM's dual-ported memory architecture. It continuously polls a given memory location and displays its contents on the screen. This polling is performed while the target is executing. If the target is writing data into the space being watched, those changes will be displayed. WATCH retrieves its port definitions and *Uni*ROM ID information from a script file.

FIND is an example of using a script file and a DOS utility program to create a utility. This example uses a batch file to invoke the script loader and then a DOS utility program. The script loader copies *Uni*ROM's memory to a DOS file. The DOS utility program then searches the file for a specified string of digits.

- find.bat invokes urload.exe and then search.exe
- find.cfg script file for find
- search.c Source code to the search routine for DOS files
- search.exe pre-compiled search.c

Documentation

This section documents the Library procedures. All Library routines share the following traits:

1. All routines are compiled in the LARGE model.
2. All routines return 0 for success, Less than 0 on error.
3. All errors codes are defined in "ecodes.h".
4. All parameters that refer to *Uni*ROM memory are absolute target addresses.
5. The *Uni*ROM library functions perform full range checking on all address parameters.
6. All structures referenced in these descriptions are defined in URLIB.H.

The Library routines are separated into three categories, PACKET ROUTINES, FILE TRANSFER ROUTINES and SCRIPT SUPPORT ROUTINES. Each are discussed below.

Packet Routines

These routines translate procedure calls to individual packets. The Library routines handle all error-checking, protocol issues and communications port details. This is an Application's main interface to UniROM.

The packet routines do NOT print to the screen. They return error codes to indicate any problems with the parameters or communications with *Uni*ROM.

The following sections discuss each routine.

SINT ur_init(void);

Initializes the communications port. It gets its parameters from the global structure "port". Ur_init must be called before calling any routines which attempt to access *Uni*ROM.

SINT ur_select(UCHAR addr);

Activates the indicated *Uni*ROM. If only one *Uni*ROM is attached, one may use the ALL-CALL ID of 0. Otherwise, the specific ID of the desired *Uni*ROM must be used. Even if there is only one *Uni*ROM attached, one must call ur_select to activate it.

SINT ur_sel_brd(UCHAR board);

Selects a specific board within 16bit *Uni*ROMs. The table below describes the valid settings for BOARD, based on the current setting for ORGANIZATION:

Organization	Valid Board Settings	Emulation Boards Selected
Single	0	Lower
Dual	0,1	Lower,Upper
Even/Odd	0,1,2	Lower,Upper,Both
Odd/Even	0,1,2	Lower,Upper,Both

*SINT ur_wt_cfg(union CONFIGS *cfg);*

Writes the configuration data in the structure pointed to by CFG to the currently selected *Uni*ROM. NOTE that this will disrupt target accesses because it re-configures the emulation hardware. One should reset the target immediately after calling this routine. Ur_wt_cfg returns non-zero on error.

*SINT ur_rd_cfg(union CONFIGS *cfg);*

Reads the configuration data of the currently selected *Uni*ROM into the structure pointed to by CFG.

*SINT ur_get_id(SCHAR *buff);*

Retrieves the currently selected *Uni*ROM's firmware version and hardware identification and writes it into the structure pointed to by BUFF. BUFF must point to a valid memory area at least 16 bytes long.

*SINT ur_rd_mem(ULONG addr,UINT count,SCHAR *buff);*

Reads COUNT bytes of data from *Uni*ROM into the buffer pointed to by BUFF. ADDR is the actual target address within *Uni*ROM of the first byte to read.

*SINT ur_wt_mem(ULONG addr,UINT count,SCHAR *data);*

Writes COUNT bytes of data from the buffer pointed to by DATA into *Uni*ROM's emulation memory. The first byte of data is written to target address ADDR within *Uni*ROM's memory space.

SINT ur_set_cntl(UCHAR num,UCHAR val);

Sets the specified control pin (1-4) on *Uni*ROM's feature connector to VAL (0 or 1).

*SINT ur_rd_status(UCHAR *buff);*

Reads the current status from *Uni*ROM and writes it into the single byte buffer pointed to by BUFF. The status is bit encoded as follows:

B7: Target Activity
B6: Target POWER ON
B5: -
B4: Target RESET active
B3: Status pin 4
B2: Status pin 3
B1: Status pin 2
B0: Status pin 1

SINT ur_reset_target(UCHAR mode);

Activates, releases or pulses the target Reset pin on *Uni*ROM's feature connector. Valid settings for MODE are:

0: Release Reset
1: Activate Reset
2: Pulse Reset active for 1/2 second and then release.

SINT ur_interrupt_target (UCHAR mode);

Activates, releases or pulses the target Interrupt pin on *Uni*ROM's feature connector. Valid settings for MODE are:

0: Release Interrupt
1: Activate Interrupt
2: Pulse Interrupt active for approximately 10us and then release.

SINT ur_enable_udx(void);

Activates a previously loaded UDX. The UDX may take over communications on this port, making it impossible to send additional packets to *Uni*ROM, or it may become a transparent extension to the binary packet protocol. The exact behavior depends on the UDX itself.

SINT ur_update_eeprom(void);

Writes the current configuration into *Uni*ROM's EEPROM.

SINT ur_setid(UCHAR id);

Changes the currently selected *Uni*ROM's ID to the specified value. Valid settings are 1 - 0xff. If an ID of 1 is used, it should ONLY be used for the LAST (or only) *Uni*ROM in the chain.

File Transfers

These procedures build on the packet commands to perform complete file transfers. All file transfer procedures use the current arbitration method to access memory unless target RESET has been asserted.

When RESET is asserted, arbitration is over-ridden, assuring the fastest possible transfers. Usually, we are not truly interested in dual-port access during complete file transfers, so we would normally assert RESET before calling one of these routines.

NOTE: The File transfer routines write messages to STDOUT using printf calls.

*SINT ur_load_bin(SCHAR *f_name,ULONG addr);*

Loads the BINARY file "F_NAME" into *Uni*ROM's emulation memory, starting at target address ADDR.

*SINT ur_verify_bin(SCHAR *f_name,ULONG addr);*

Verifies *Uni*ROM's emulation memory starting at target address ADDR against the contents of the BINARY file "F_NAME".

*SINT ur_save_bin(SCHAR *f_name,ULONG start,ULONG end);*

Saves a range of *Uni*ROM's Emulation Memory to a file. START and END specify the addresses of the FIRST and LAST bytes of the range to transfer.

SINT ur_fill(ULONG start,ULONG end,UCHAR val);

Fills a range of *Uni*ROM's Emulation Memory with the character VAL. START specifies the address of the first byte to fill. END specifies the address of the LAST byte to fill.

*SINT ur_load_udx(SCHAR *f_name);*

Loads the specified file into *Uni*ROM's UDX memory space and verifies it.

*SINT ur_load_hex(SCHAR *file_name,ULONG addr);*

Loads the HEX file "F_NAME" into *Uni*ROM's emulation memory. ADDR is a default address used only if the HEX file does not contain full address information. Most HEX files will provide adequate address information to allow us to properly locate the file within *Uni*ROM's address space. The ADDR parameter is used primarily to allow 16 bit addressing mode HEX files to locate into a larger address space. When used, ADDR will usually be set to the starting address of the EPROM or the starting address of the code image.

*SINT ur_verify_hex(SCHAR *file_name,ULONG addr);*

Verifies the HEX file "F_NAME" against the data in UniROM the HEX file "F_NAME" against the data in *Uni*ROM's emulation memory. See the previous command for a description of the use of the ADDR parameter.

Script File Support

These procedures let the user parse *Uni*ROM SCRIPT files.

Get_port_cfg and get_cfg_section use “parse_print” to display any error messages. Parse_script_cmd writes to STDOUT using printf calls.

*SINT get_port_cfg(FILE *fp);*

Parses the script file pointed to by FP for the port definition. If a port definition is found, it parses the information into the global structure “port” and returns a 0. If it hits the end-of-file or a CONFIGURATION section or a COMMAND section without finding a port definition, it returns non-zero.

*SINT get_cfg_section(FILE *fp,union CONFIGS *cfg);*

Finds and parses the next CONFIGURATION section in the script file pointed to by FP. The parsed configuration is written into the structure pointed to by CFG.

Returns non-zero when it hits the end-of-file or the COMMAND section.

*UCHAR parse_script_cmd(FILE *fp,UCHAR trace_flag);*

This single command parses and executes ALL commands in the COMMAND section of the script file. Note that this routine will write status information to the screen with printf statements.

*void parse_print(SCHAR ecode,UINT lnum,SCHAR *lp);*

Parse_print prints all error messages from “get_port_cfg” and “get_cfg_section” to the standard output screen. One can replace this routine with a custom version to re-direct error messages to another destination, like a windowing routine library.

Ecode is the error code returned by the parser. Lnum is the line number in the script file that contains the error. *LP is a pointer to a string variable that holds a copy of the line.