# Table of Contents

TABLE OF CONTENTS .................................................................................. 2

1 OVERVIEW .......................................................................................... 3

2 SYSTEM FUNCTIONS ........................................................................... 3

2.1 TASK MANAGEMENT FUNCTIONS .................................................. 3
2.2 TASK DEPENDENT SYNCHRONIZATION FUNCTIONS ...................... 3
2.3 TASK EXCEPTION HANDLING FUNCTIONS ..................................... 4
2.4 SYNCHRONIZATION AND COMMUNICATION FUNCTIONS ............. 4
2.5 EXTENDED SYNCHRONIZATION AND COMMUNICATION FUNCTIONS .... 5
2.6 MEMORY POOL MANAGEMENT FUNCTIONS ................................... 6
2.7 TIME MANAGEMENT FUNCTIONS .................................................. 6
2.8 SYSTEM STATE MANAGEMENT FUNCTIONS ................................... 7
2.9 INTERRUPT MANAGEMENT FUNCTIONS ....................................... 7
2.10 SERVICE CALL MANAGEMENT FUNCTIONS .................................. 7
2.11 SYSTEM CONFIGURATION MANAGEMENT FUNCTIONS .................. 7

3 BUILDING THE DEVELOPMENT ENVIRONMENT ..................................... 8

3.1 THE HARDWARE PLATFORM ............................................................ 8
    3.1.1 The HOST ........................................................................... 8
    3.1.2 The TARGET ....................................................................... 8
3.2 THE SOFTWARE ENVIRONMENT ....................................................... 9

4 HOW TO PROGRAM, BUILD AND DEBUG APPLICATIONS ...................... 9

4.1 PROGRAM APPLICATIONS ............................................................... 9
4.2 BUILD APPLICATIONS .................................................................... 11
4.3 DEBUG APPLICATIONS .................................................................... 11

5 ADVANCED ISSUES ............................................................................. 12

5.1 CONFIGURATION ON DIFFERENT PLATFORMS ................................. 12
5.2 INSTALLING THE EXCEPTION HANDLER AND INTERRUPT HANDLER ...... 13
    5.2.1 Installing the Exception Handler .......................................... 13
    5.2.2 Installing the Interrupt Handler ......................................... 14
    5.2.3 Boot ............................................................................... 15

6 APPENDIX ......................................................................................... 18

6.1 TESTED BOARDS LIST ................................................................. 18
6.2 EXAMPLE PLATFORM MAP ............................................................ 18
6.3 Initializing the Memory Component .................................................. 18
6.4 How to Program the OS Image (jsp.bin) and the Boot-Loader Image     18
    boot.bin to the Flash .................................................................. 18
6.5 Link File for Store and Run the Image in the Internal Memory .......... 20
6.6 Link File for Store the Image in Flash and Run in the SRAM .......... 23
1 Overview

The user manual describes how to develop applications on the JSP kernel for the LatticeMico32. The TOPPERS/JSP is a royalty-free, open source embedded real-time operation system, developed by the TOPPERS Project. JSP is an acronym for Just Standard Profile, and as the name shows, is implemented in accordance with the µITRON4.0 specification standard profile regulation. The LatticeMico32™ is a configurable 32-bit soft processor core for Lattice Field Programmable Gate Array (FPGA) devices. By combining a 32-bit wide instruction set with 32 general purpose registers, the LatticeMico32 provides the performance and flexibility suitable for a wide variety of markets, including communications, consumer, computer, medical, industrial, and automotive. With separate instruction and data buses, this Harvard architecture processor allows for single-cycle instruction execution as the instruction and data memories can be accessed simultaneously. Additionally, the LatticeMico32 uses a Reduced Instruction Set Computer (RISC) architecture; thereby, providing a simpler instruction set and faster performance.

This manual is targeted to the software programmers who are interested in developing JSP applications for the LatticeMico32.

2 System Functions

The JSP system functions are listed in the following tables\(^1\). And C library is not be implemented.

2.1 Task Management Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRE_TSK</td>
<td>Create Task</td>
<td>Static</td>
<td>Yes</td>
</tr>
<tr>
<td>cre_tsk</td>
<td>Create Task</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>acre_tsk</td>
<td>Create Task (ID Number Automatic Assignment)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>del_tsk</td>
<td>Delete Task</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>act_tsk</td>
<td>Activate Task</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>lact_tsk</td>
<td>Activate Task</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>can_act</td>
<td>Cancel Task Activation Requests</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>sta_tsk</td>
<td>Activate Task (with a Start Code)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>ext_tsk</td>
<td>Terminate Invoking Task</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>exd_tsk</td>
<td>Terminate and Delete Invoking Task</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>ter_tsk</td>
<td>Terminate Task</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>chg_pri</td>
<td>Change Task Priority</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>get_pri</td>
<td>Reference Task Priority</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>ref_tsk</td>
<td>Reference Task State</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>ref_tst</td>
<td>Reference Task State (Simplified Version)</td>
<td>C Language</td>
<td>No</td>
</tr>
</tbody>
</table>

2.2 Task Dependent Synchronization Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>slp_tsk</td>
<td>Put Task to Sleep</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>tslp_tsk</td>
<td>Put Task to Sleep (with Timeout)</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>wup_tsk</td>
<td>Wakeup Task</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>iwup_tsk</td>
<td>Wakeup Task</td>
<td>C Language</td>
<td>Yes</td>
</tr>
</tbody>
</table>

\(^1\) For details on system functions, refer to the “µITRON4.0 Specification”. In the JSP, the functions defined in the “µITRON4.0 Specification” are implemented.)

March, 2008
### 2.3 Task Exception Handling Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF_TEX</td>
<td>Define Task Exception Handling Routine</td>
<td>Static</td>
<td>Yes</td>
</tr>
<tr>
<td>def_tex</td>
<td>Define Task Exception Handling Routine</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>ras_tex</td>
<td>Raise Task Exception Handling</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>iras_tex</td>
<td>Raise Task Exception Handling</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>dis_tex</td>
<td>Disable Task Exceptions</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>ena_tex</td>
<td>Enable Task Exceptions</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>sns_tex</td>
<td>Reference Task Exception Handling State</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>ref_tex</td>
<td>Reference Task Exception Handling State</td>
<td>C Language</td>
<td>No</td>
</tr>
</tbody>
</table>

### 2.4 Synchronization and Communication Functions

#### Semaphores

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRE_SEM</td>
<td>Create Semaphore</td>
<td>Static</td>
<td>Yes</td>
</tr>
<tr>
<td>cre_sem</td>
<td>Create Semaphore</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>acre_sem</td>
<td>Create Semaphore (ID Number Automatic Assignment)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>del_sem</td>
<td>Delete Semaphore</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>sig_sem</td>
<td>Release Semaphore Resource</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>isig_sem</td>
<td>Release Semaphore Resource</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>wai_sem</td>
<td>Acquire Semaphore Resource</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>pol_sem</td>
<td>Acquire Semaphore Resource (Polling)</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>twai_sem</td>
<td>Acquire Semaphore Resource (with Timeout)</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>ref_sem</td>
<td>Reference Semaphore State</td>
<td>C Language</td>
<td>No</td>
</tr>
</tbody>
</table>

#### Eventflags

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRE_FLG</td>
<td>Create Eventflag</td>
<td>Static</td>
<td>Yes</td>
</tr>
<tr>
<td>cre_flg</td>
<td>Create Eventflag</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>acre_flg</td>
<td>Create Eventflag (ID Number Automatic Assignment)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>del_flg</td>
<td>Delete Eventflag</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>set_flg</td>
<td>Set Eventflag</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>iset_flg</td>
<td>Set Eventflag</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>clr_flg</td>
<td>Clear Eventflag</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>wai_flg</td>
<td>Wait for Eventflag</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>pol_flg</td>
<td>Wait for Eventflag (Polling)</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>twai_flg</td>
<td>Wait for Eventflag (with Timeout)</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>ref_flg</td>
<td>Reference Eventflag Status</td>
<td>C Language</td>
<td>No</td>
</tr>
</tbody>
</table>

#### Data Queues

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRE_DTQ</td>
<td>Create Data Queue</td>
<td>Static</td>
<td>Yes</td>
</tr>
<tr>
<td>cre_dtq</td>
<td>Create Data Queue</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>acre_dtq</td>
<td>Create Data Queue (ID Number Automatic Assignment)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>del_dtq</td>
<td>Delete Data Queue</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>snd_dtq</td>
<td>Send to Data Queue</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>psnd_dtq</td>
<td>Send to Data Queue (Polling)</td>
<td>C Language</td>
<td>Yes</td>
</tr>
</tbody>
</table>
ipsnd_dtq  Send to Data Queue (Polling)  C Language  Yes
snd_dtq   Send to Data Queue (with Timeout)  C Language  Yes
fsnd_dtq  Forced Send to Data Queue  C Language  Yes
ifsnd_dtq Forced Send to Data Queue  C Language  Yes
crv_dtq   Receive from Data Queue  C Language  Yes
prcv_dtq  Receive from Data Queue (Polling)  C Language  Yes
trcv_dtq  Receive from Data Queue (with Timeout)  C Language  Yes
ref_dtq   Reference Data Queue State  C Language  No

Mailboxes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRE_MBX</td>
<td>Create Mailbox</td>
<td>Static</td>
<td>Yes</td>
</tr>
<tr>
<td>cre_mbx</td>
<td>Create Mailbox</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>acre_mbx</td>
<td>Create Mailbox (ID Number Automatic Assignment)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>del_mbx</td>
<td>Delete Mailbox</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>snd_mbx</td>
<td>Send to Mailbox</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>rcv_mbx</td>
<td>Receive from Mailbox</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>prcv_mbx</td>
<td>Receive from Mailbox (Polling)</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>trcv_mbx</td>
<td>Receive from Mailbox (with Timeout)</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>ref_mbx</td>
<td>Reference Mailbox State</td>
<td>C Language</td>
<td>No</td>
</tr>
</tbody>
</table>

2.5 Extended Synchronization and Communication Functions

Mutexes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRE_MTX</td>
<td>Create Mutex</td>
<td>Static</td>
<td>No</td>
</tr>
<tr>
<td>cre_mtx</td>
<td>Create Mutex</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>acre_mtx</td>
<td>Create Mutex (ID Number Automatic Assignment)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>del_mtx</td>
<td>Delete Mutex</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>loc_mtx</td>
<td>Lock Mutex</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>ploc_mtx</td>
<td>Lock Mutex (Polling)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>tloc_mtx</td>
<td>Lock Mutex (with Timeout)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>unl_mtx</td>
<td>Unlock Mutex</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>ref_mtx</td>
<td>Reference Mutex State</td>
<td>C Language</td>
<td>No</td>
</tr>
</tbody>
</table>

Message Buffers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRE_MBF</td>
<td>Create Message Buffer</td>
<td>Static</td>
<td>No</td>
</tr>
<tr>
<td>cre_mbf</td>
<td>Create Message Buffer</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>acre_mbf</td>
<td>Create Message Buffer (ID Number Automatic Assignment)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>del_mbf</td>
<td>Delete Message Buffer</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>snd_mbf</td>
<td>Send to Message Buffer</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>psnd_mbf</td>
<td>Send to Message Buffer (Polling)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>tsnd_mbf</td>
<td>Send to Message Buffer (with Timeout)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>rcv_mbf</td>
<td>Receive from Message Buffer</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>prcv_mbf</td>
<td>Receive from Message Buffer (Polling)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>trcv_mbf</td>
<td>Receive from Message Buffer (with Timeout)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>ref_mbf</td>
<td>Reference Message Buffer State</td>
<td>C Language</td>
<td>No</td>
</tr>
</tbody>
</table>

Rendezvous

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRE_POR</td>
<td>Create Rendezvous Port</td>
<td>Static</td>
<td>No</td>
</tr>
<tr>
<td>cre_por</td>
<td>Create Rendezvous Port</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>acre_por</td>
<td>Create Rendezvous Port (ID Number Automatic Assignment)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>del_por</td>
<td>Delete Rendezvous Port</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>cal_por</td>
<td>Call Rendezvous Port</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>tcal_por</td>
<td>Call Rendezvous Port (with Timeout)</td>
<td>C Language</td>
<td>No</td>
</tr>
</tbody>
</table>
### 2.6 Memory Pool Management Functions

#### Fixed-Sized Memory Pools

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRE_MPF</td>
<td>Create Fixed-Sized Memory Pool</td>
<td>Static</td>
<td>Yes</td>
</tr>
<tr>
<td>cre_mpf</td>
<td>Create Fixed-Sized Memory Pool</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>acre_mpf</td>
<td>Create Fixed-Sized Memory Pool (ID Number Automatic Assignment)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>del_mpf</td>
<td>Delete Fixed-Sized Memory Pool</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>get_mpf</td>
<td>Acquire Fixed-Sized Memory Block</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>pget_mpf</td>
<td>Acquire Fixed-Sized Memory Block (Polling)</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>tget_mpf</td>
<td>Acquire Fixed-Sized Memory Block (with Timeout)</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>rel_mpf</td>
<td>Release Fixed-Sized Memory Block</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>ref_mpf</td>
<td>Reference Fixed-Sized Memory Pool State</td>
<td>C Language</td>
<td>No</td>
</tr>
</tbody>
</table>

#### Variable-Sized Memory Pools

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRE_MPL</td>
<td>Create Variable-Sized Memory Pool</td>
<td>Static</td>
<td>No</td>
</tr>
<tr>
<td>cre_mpl</td>
<td>Create Variable-Sized Memory Pool</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>acre_mpl</td>
<td>Create Variable-Sized Memory Pool (ID Number Automatic Assignment)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>del_mpl</td>
<td>Delete Variable-Sized Memory Pool</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>get_mpl</td>
<td>Acquire Variable-Sized Memory Block</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>pget_mpl</td>
<td>Acquire Variable-Sized Memory Block (Polling)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>tget_mpl</td>
<td>Acquire Variable-Sized Memory Block (with Polling)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>rel_mpl</td>
<td>Release Variable-Sized Memory Block</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>ref_mpl</td>
<td>Reference Variable-Sized Memory Pool State</td>
<td>C Language</td>
<td>No</td>
</tr>
</tbody>
</table>

### 2.7 Time Management Functions

#### System Time Management

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_tim</td>
<td>Set System Time</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>get_tim</td>
<td>Reference System Time</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>isig_tim</td>
<td>Supply Time Tick</td>
<td>C Language</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### Cyclic Handlers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRE_CYC</td>
<td>Create Cyclic Handler</td>
<td>Static</td>
<td>Yes</td>
</tr>
<tr>
<td>cre_cyc</td>
<td>Create Cyclic Handler</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>acre_cyc</td>
<td>Create Cyclic Handler (ID Number Automatic Assignment)</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>del_cyc</td>
<td>Delete Cyclic Handler</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>sta_cyc</td>
<td>Start Cyclic Handler Operation</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>stp_cyc</td>
<td>Stop Cyclic Handler Operation</td>
<td>C Language</td>
<td>Yes</td>
</tr>
<tr>
<td>ref_cyc</td>
<td>Reference Cyclic Handler Operation</td>
<td>C Language</td>
<td>No</td>
</tr>
</tbody>
</table>

#### Alarm Handlers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRE_ALM</td>
<td>Create Alarm Handler</td>
<td>Static</td>
<td>No</td>
</tr>
</tbody>
</table>
Create Alarm Handler (ID Number Automatic Assignment)
Create Alarm Handler
Delete Alarm Handler
Start Alarm Handler Operation
Stop Alarm Handler Operation
Reference Alarm Handler State

Delete Overrun Handler
Define Overrun Handler
Start Overrun Handler Operation
Stop Overrun Handler Operation
Reference Overrun Handler State

Overrun Handlers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF_OVR</td>
<td>Define Overrun Handler</td>
<td>Static</td>
<td>No</td>
</tr>
<tr>
<td>def_ovr</td>
<td>Define Overrun Handler</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>sta_ovr</td>
<td>Start Overrun Handler Operation</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>stp_ovr</td>
<td>Stop Overrun Handler Operation</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>ref_ovr</td>
<td>Reference Overrun Handler State</td>
<td>C Language</td>
<td>No</td>
</tr>
</tbody>
</table>

Rotate Task Precedence
Rotate Task Precedence
Reference Task ID in the RUNNING State
Reference Task ID in the RUNNING State
Lock the CPU
Lock the CPU
Unlock the CPU
Unlock the CPU
Disable Dispatching
Enable Dispatching
Reference Contexts
Reference CPU State
Reference Dispatching State
Reference Dispatching Pending State
Reference System State

2.8 System State Management Functions

2.9 Interrupt Management Functions

Define Interrupt Handler
Define Interrupt Handler
Attach Interrupt Service Routine
Create Interrupt Service Routine
Create Interrupt Service Routine (ID Number Automatic Assignment)
Delete Interrupt Service Routine
Reference Interrupt Service Routine State
Disable Interrupt
Enable Interrupt
Change Interrupt Mask
Reference Interrupt Mask

2.10 Service Call Management Functions

Define Extended Service Call
Define Extended Service Call
Invoke Service Call

2.11 System Configuration Management Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>API</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF_SVC</td>
<td>Define Extended Service Call</td>
<td>Static</td>
<td>No</td>
</tr>
<tr>
<td>def_svc</td>
<td>Define Extended Service Call</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>cal_svc</td>
<td>Invoke Service Call</td>
<td>C Language</td>
<td>No</td>
</tr>
</tbody>
</table>

March, 2008
3 Building the Development Environment

3.1 The Hardware Platform

Before you can build applications, the hardware platform must be established. In general, the embedded hardware platform composes of two parts: the HOST and the TARGET. The following diagram illustrates the basic composition of the hardware platform.

![Diagram of System Development Platform]

3.1.1 The HOST

Here are the steps that needs to be accomplished on the host:
1. Build the JSP kernel image for the target LatticeMico32 Development Board.
2. Download the JSP kernel image to the target board through the JTAG port.
3. Observe that the debug information are displayed on the target board through the UART port.

3.1.2 The TARGET

On the target side, run the image. Send the information to the host or receive the input from the host through the UART port. Make sure that the following components are included in the target board:
- Timer
- UART
- Storage Component:
  - Internal RAM
  - External SRAM
  - FLASH

There are two possible configurations of the storage component. If the internal RAM is large enough to accommodate the kernel image, the storage component will be the internal RAM only. Otherwise the image must be stored in the flash and run in an external SRAM. In this case, the storage component must include the external SRAM and the flash.2

---


---

<table>
<thead>
<tr>
<th>DEF_EXC</th>
<th>Define CPU Exception Handler</th>
<th>Static</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>def_exc</td>
<td>Define CPU Exception Handler</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>ref_cfg</td>
<td>Reference Configuration Information</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>ref_ver</td>
<td>Reference Version Information</td>
<td>C Language</td>
<td>No</td>
</tr>
<tr>
<td>ATT_INI</td>
<td>Attach Initialization Routine</td>
<td>Static</td>
<td>Yes</td>
</tr>
</tbody>
</table>
3.2 The Software Environment

Before you can build applications, the software environment must be ready. Here is a checklist for you to check the readiness of your software environment:

1. The LatticeMico32 Development Tools are installed.
2. The JSP source code and the hardware-dependence code for LatticeMico32 are available.
3. The hardware-dependent code for LatticeMico32 is placed in the directory “jsp\config”.
4. The C compiler is installed. It can be one among the following three:
   - Microsoft Visual C++ 6.0
   - GCC
   - BCC
5. The configurator is built. If not, follow the next steps to build the configurator:
   - If the Microsoft Visual C++ 6.0 is installed
     1. Enter the directory “jsp\cfg\vc_project”.
     2. Open the file “configurator.dsw” with the Microsoft Visual C++ 6.0.
     3. Click the menu command “build” and select the option “Batch Build…”.
     4. Select all projects to be built. When completed, the file “cfg.exe” is generated in the directory “jsp\cfg”.
   - If the GCC is installed
     1. Enter the directory “jsp\cfg”
     2. Run the file “Makefile”
   - If the BCC is installed
     1. Enter the directory “jsp\cfg”
     2. Run the file “Makefile.bcc”

4 How to Program, Build and Debug Applications

4.1 Program Applications

The Application source code compose of two parts:

1. The configuration file (*.cfg)
2. The C file (*.c) and the include file(*.h).

The file names of the C file and the configuration file must be the same.

In the configuration file, the function can only be invoked with “API” that is marked “Static” in the function tables introduced in the previous sections.

Configuration information is usually defined in the file and includes the following sections:

- Task ID
- Task Attribute
- Task Start address
- Task Initial Priority
- Task Stack Size
- Exception Handler
- Interrupt Handler

In the C file, you can write the main body of those tasks that are defined in the configuration file.

The macro used for the configuration file is defined in the include file. Also the function definition about the task must be included in the include file. Insert the definition of the task function between the line “#ifndef _MACRO_ONLY” and the line “#endif” in the include file.

The following example helps you understand it better. The example, when implemented, prints the string “Hello World!!!” to the UART every 2 seconds.

To implement the example, follow these steps:

Step 1: Create a configuration file with the name “hello_world.cfg”. Copy the following content into the configuration file:

```c
/*-----------------------------hello_world.cfg----- ------------------*/
#define _MACRO_ONLY
#include "hello_world.h"

#include "hello_world.h"
CRE_TSK(TASK_ID, { TA_HLNG|TA_ACT, NULL, hello_world_task, TASK_PRIORITY,
STACK_SIZE, NULL });
#include "./systask/serial.cfg"
#include "./systask/serial.cfg"

/*----------------------------------------------------------------------------*/

Step 2: Create the file “hello_world.c” and copy the following content into the file. This file defines the main body of the task.:

```c
/*---------------------------hello_world.c--------- ------------------*/
#include <t_services.h>
#include "../kernel/jsp_kernel.h"

void hello_world_task (VP_INT exinf)
{
    for(;;)
    {
        syslog_printf("Hello World!!!\n", NULL, sys_putc);
        syscall(dly_tsk(2));
    }

/*----------------------------------------------------------------------------*/

Step 3: Create the file “hello_world.h” and copy the following content into the file.

```c
/*---------------------------hello_world.h--------- ------------------*/
#ifndef __HELLO_WORLD_H__
#define __HELLO_WORLD_H__

#define TASK_PRIORITY 5
#define STACK_SIZE 512

#ifndef _MACRO_ONLY
extern void hello_world_task(VP_INT exinf);

/*----------------------------------------------------------------------------*/

Step 3: Create the file “hello_world.h” and copy the following content into the file.

```c
/*---------------------------hello_world.h--------- ------------------*/
#ifndef __HELLO_WORLD_H__
#define __HELLO_WORLD_H__

#define TASK_PRIORITY 5
#define STACK_SIZE 512

#ifndef _MACRO_ONLY
extern void hello_world_task(VP_INT exinf);

/*----------------------------------------------------------------------------*/

The macro “TASK_ID” does not need to be defined. It will be generated automatically in the build process. The macro “TA_HLNG” is defined in the file “jsp\include\kernel.h”. Refer to the “uITRON4.0 Specification” for more information about the task macro definition.
4.2 Build Applications

Before build, the link file “config\lm32\ECP2\lm32elf.ld” should be ready, it decided by the image storage format. Appendix 6.5 and 6.6 lists two kinds of link file for the “hello_world” example, the following steps show you how to build applications.

Step 1: Copy the C file, include file and the configuration file to the destination directory. The destination directory can be set as the “jsp” or the subdirectory under the “jsp”, such as “jsp/test”.

Step 2: Open the LatticeMico32 System Shell and enter the destination directory in the shell.

Step 3: Run the following command in the shell:

If the destination directory is “jsp”,

Command: ./configure -C lm32 -S ECP2 -A hello_world

If the destination directory is “jsp/test”

The related directory in the application code should be modified accordingly.

Command: ../configure –C lm32 –S ECP2 –A hello_world.

Step 4: Run the following command in the shell:

make depend

Five files (kernel_id.h, kernel_cfg.c, kernel_chk.c, Makefile.depend and offset.h) will be generated.

Step 5: Run the following command in the shell:

make; make jsp.bin

This step generates the file “jsp.exe” and “jsp.bin”

If some modification was made to the application code, please go to run Step 4 and Step 5.

4.3 Debug Applications

The following steps show you how to debug applications:

Step 1: Open the LatticeMico32 System shell and type the following command in the shell

TCP2JTAGVC2

Step 2: Open another LatticeMico32 System shell and enter the destination directory, type the following command to enter the GDB debug environment:

lm32-elf-gdb jsp.exe

Step 3: Enter the GDB debug environment and link to the target board by entering the following command in the GDB debug environment:
Step 4: Load the image to the target board by entering the following command in the GDB debug environment:

```
load
```

Step 5: The application can be debugged now. Refer to the GDB User Manual for details of the GDB usage.

5 Advanced Issues

5.1 Configuration on Different Platforms

Because the SOC (System on Chip) can be modified, the platform may accordingly have different configurations. Here is a list of possible different configurations:
1. The clock of the target CPU is variable.
2. The target board UART baud rate used to communicate with the PC is different from the default.
3. The UART registers start address or Timer registers start address of the target board is different from the default.
4. The target board SRAM start address or size is different from the default.

The configuration of the target board can be found in the header file “jsp\config\lm32\ECP2\lm32.h”. Here is an example of the header file:

```
/*------------------------- lm32.h --------------------------*/
......
#define MICO32_CPU_CLOCK_HZ     (25000000)
#define DEFAULT_UART_BAUDRATE   (115200)
#define TIMER_BASE_REG          (0x80000100)
#define UART1_BASE_REG          (0x80000180)
......
/*------------------------- lm32.h --------------------------*/
```

When the following configuration of the target board is different from the default, you can modify the corresponding definition in the file “lm32.h”:
- CPU Clock
- UART Baud rate
- UART Registers Start Address
- Timer Registers Start Address

If the SRAM start address or size needs to be modified, you can edit the link file “jsp\config\lm32\ECP2\lm32elf.ld”. Here is an example of the link file:

```
/*--------------------- lm32elf.ld --------------------------*/
MEMORY
|
| sram  : ORIGIN = 0x00100000, LENGTH = 131072
|
|)
......
/*--------------------- lm32elf.ld --------------------------*/
```
ORIGIN = 0xXXXXXXXX
Length = YYYYYYYY
XXXXXXXX : The SRAM Start Address
YYYYYYYY : The SRAM Size

5.2 Installing the Exception Handler and Interrupt Handler

5.2.1 Installing the Exception Handler
In the JSP, there are two types of exception handlers:
1. System exception handler
2. Task exception handler

The following diagram lists the system exceptions in the LatticeMico32

<table>
<thead>
<tr>
<th>Exception</th>
<th>ID</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>0</td>
<td>Raised when the processor’s reset pin is asserted.</td>
</tr>
<tr>
<td>Breakpoint</td>
<td>1</td>
<td>Raised when either a break instruction is executed or when a hardware breakpoint is triggered.</td>
</tr>
<tr>
<td>InstructionBusError</td>
<td>2</td>
<td>Raised when an instruction fetch fails, typically due to the requested address being invalid.</td>
</tr>
<tr>
<td>Watchpoint</td>
<td>3</td>
<td>Raised when a hardware watchpoint is triggered.</td>
</tr>
<tr>
<td>DataBusError</td>
<td>4</td>
<td>Raised when a data access fails, typically due to either the requested address being invalid or the type of access not being allowed.</td>
</tr>
<tr>
<td>DivideByZero</td>
<td>5</td>
<td>Raised when an attempt is made to divide by zero.</td>
</tr>
<tr>
<td>Interrupt</td>
<td>6</td>
<td>Raised when one of the processor’s interrupt pins is asserted, providing that the corresponding field in the interrupt mask (IM) CSR is set and the global interrupt enable flag, IE.IE, is set.</td>
</tr>
<tr>
<td>SystemCall</td>
<td>7</td>
<td>Raised when a scall instruction is executed.</td>
</tr>
</tbody>
</table>

Note: Users cannot install the system interrupt exception handler. It will be installed automatically. If the user attempts to replace the system interrupt exception handler with a customized handler, it will prevent the system interrupt handler to be properly installed.
The following diagram describes the relationship between the system exception handler and task exception handler.

[Diagram showing the relationship between system and task exception handlers]

The system exception handler is only one to handle all system exceptions. Whereas each task owns an independent task exception handler. When an exception occurs during task running, the system exception handler be executed first. After the system exception handler has finished execution, the task exception handler of the current running task starts to run.

For the JSP, add the following definition in the configuration file to install the system exception handler:

\[
\text{DEF_EXC(SYS_EXC_ID, \{ TA_HLNG, sys_exc_handler \});}
\]

SYS_EXC_ID : System Exception ID
Sys_exc_handler : System exception handler name that must be implemented in the C file.

Add the following definition in the configuration file to attach the task exception handler to the task:

\[
\text{DEF_TEX(TASK_ID, \{ TA_HLNG, tsk_exc_routine \});}
\]

TASK_ID : Task ID attached
tsk_exc_routine : Task exception handler name that must be implemented in the C file.

5.2.2 Installing the Interrupt Handler
In the LatticeMico32 system, the maximum interrupt number is 32.
In the JSP for LatticeMico32, the timer and UART have occupied two interrupt numbers. Therefore, the interrupt number left for the user is 30.

The default interrupt number of the timer is 0.
The default interrupt number of the UART is 1.

In the JSP for LatticeMico32, the interrupt handler of low priority tasks can be preempted by that of a high priority task. Smaller interrupt number means higher interrupt priority.

Add the following definition in the configuration file to install the system exception handler:

```c
DEF_INH(INHNO_DEV, { TA_HLNG, dev_intr_handler });
```

**INHNO_DEV** : Device Interrupt Number  
**dev_intr_routine** : Interrupt handler for the device

### 5.2.3 Boot

As mentioned before, the storage component can be one of the following components:

1. **Internal RAM only**
   
   In this case, before building the images, the starting address and size in the link file should be modified, (details please refer to the Appendix6.5), then build the OS image refer to the Section 4.2 Build Applications. About how to program the binary file to the Internal RAM refer the following example steps:
   
   Step1: Open the LatticeMico32 System Cygwin Shell and enter the test destination directory.
   
   Step2: Run the following command in the shell:
   
   ```bash
   $ bin_to_verilog --EB --width 4 jsp.bin EBR.mem
   ```
   
   Step3: Use the generated memory initialization file EBR.mem to initialize EBR in FPGA. Details see the Appendix6.3.
   
   Step4: Use the ispVM download the generated bitstream to the FPGA, then the application is working.

2. **Flash + External SRAM**

   In this case, you need to program a boot-loader that loads the image from the flash to the external SRAM.
The following example steps describe the flow facility.

**Figure 3: Image Distribution Chart.**

<table>
<thead>
<tr>
<th>OS Image (Run)</th>
<th>0x02000000</th>
<th>Externel SRAM (0x2000000 – 0x20FFFFF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>jsp.bin</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boot-loader</th>
<th>0x020FFFFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot.bin</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OS Image (Store)</th>
<th>0x04000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>jsp.bin</td>
<td></td>
</tr>
</tbody>
</table>

| 0x041FFFFF      |            | Flash (0x4000000 – 0x41FFFFF) |
|----------------|------------|

| 0xFFFFFFFF      |            |                                  |

---

**Step 1:** Build the OS image. Because the OS image runs in the external SRAM.

The start address and length in the link file “config\lm32\ECP2\lm32elf.ld” should be equal to that of the external SRAM. Details refer to the Appendix 6.6. About how to build the OS image, refer to the Section 4.2 Build Applications.

**Step 2:** Program a boot-loader that copy the OS image from the flash to the external SRAM. After copying is complete, the boot-loader jumps to the external SRAM and continue to run.

The boot-loader source code can be found in the directory “src\boot\”. In the header file “boot.h”, the parameters are defined.

**SRAM_START_ADDRESS**

The SRAM start address. In the above example, it’s 0x02000000.

**SRAM_SIZE**

The size of the SRAM. In the above example, it’s 0x00100000.

**FLASH_OS_ADDRESS**

The start address where the OS image was stored in the flash. The above example, it’s 0x04080000.

**OS_IMAGE_SIZE**

The size of the OS image. It can be gotten from the OS image file.
The OS image (jsp.bin) was stored from the address 0x4000000 to 0x41FFFFF. If the flash address or size is changed, the corresponding link file (src\boot\boot.ld) need to be modified.

About how to build the boot-loader image, you can run the “LatticeMico32 System SDK Shell” firstly. Then enter the directory “src\boot” and type the command “make” in the shell, then boot.bin generates under the src\boot.

Step 3: Program the boot-loader image (boot.bin) and the OS image (jsp.bin) to the flash, Details please refer to Appendix6.4.

Step 4: After successfully write the boot.bin and jsp.bin to the flash, open the Hyper Terminal and set it correctly, then Reset or re-power_up the evaluation board, the application is working.
6 Appendix

6.1 Tested Boards List
The porting JSP was tested on the following three board, details please refer to
the release package JSP for lm32 v1.2/test/
- ECP2 LatticeMico32/DSP development board
- ECP2M PCI express platform evaluation board
- LatticeSCTM communications platform evaluation board

6.2 Example Platform Map
The example platform is based on the LatticeMico32/DSP Development Board
(ECP2). Refer to the following tables for settings of the platform memory map:

<table>
<thead>
<tr>
<th>Component</th>
<th>From</th>
<th>End</th>
<th>Size (byte)</th>
<th>Interrupt Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Sram</td>
<td>0x02000000</td>
<td>0x020FFFFF</td>
<td>0x00100000</td>
<td>N/A</td>
</tr>
<tr>
<td>Flash</td>
<td>0x04000000</td>
<td>0x05FFFFF</td>
<td>0x02000000</td>
<td>N/A</td>
</tr>
<tr>
<td>Timer</td>
<td>0x80000100</td>
<td>0x800001FF</td>
<td>0x00000080</td>
<td>0</td>
</tr>
<tr>
<td>UART</td>
<td>0x80000180</td>
<td>0x800001FF</td>
<td>0x00000080</td>
<td>1</td>
</tr>
</tbody>
</table>

Boot from Internal RAM (EBR):

<table>
<thead>
<tr>
<th>Component</th>
<th>From</th>
<th>End</th>
<th>Size (byte)</th>
<th>Interrupt Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBR</td>
<td>0x00100000</td>
<td>0x0010FFFF</td>
<td>0x00008000</td>
<td>N/A</td>
</tr>
<tr>
<td>External Sram</td>
<td>0x00200000</td>
<td>0x002FFFFF</td>
<td>0x00100000</td>
<td>N/A</td>
</tr>
<tr>
<td>Timer</td>
<td>0x80000100</td>
<td>0x800001FF</td>
<td>0x00000080</td>
<td>0</td>
</tr>
<tr>
<td>UART</td>
<td>0x80000180</td>
<td>0x800001FF</td>
<td>0x00000080</td>
<td>1</td>
</tr>
</tbody>
</table>

6.3 Initializing the Memory Component
Now you load the memory initialization file into a placed and routed FPGA bitstream.
To implement the .mem file in an ispLEVER design:
1. In ispLEVER, double-click Place & Route Design.
2. In the Project Navigator toolbar, choose Tools > Memory Initialization Tool or
   click on double-click Memory Initialization in the Processes for Current Source
   window of the Project Navigator.
3. Click Load, browse to the .ncd file or type its name in the File Name box, and click
   Open.
4. In the EBR Memories in Design panel, select \{OCM\}/ram, where \{OCM\} is the
   name of the on-chip memory component from the LatticeMico32 platform.
5. Ensure that Memory Format is set to HEX.
6. In the Memory File box, browse to and select the on-chip memory initialization
   (.mem) file created by LatticeMico32.
7. Click Apply Change.
8. Click Save, and dismiss the Save Design panel.
9. Click Exit.
10. In the Project Navigator, double-click Generate Bitstream Data.

6.4 How to Program the OS image (jsp.bin) and the boot-loader image
boot.bin to the flash.
Using the example project “CFIFlashProgrammer” include in the
LatticeMico32 system to write the flash. Following is the steps:

---

5 For details about the board, refers to “LatticeMico32/DSP Development board users guide” and
   “LatticeMico32/DSP Development Kit user’s Guide for LatticeECP2”.
Step 1. Open the mico32 development system and build up one new project by click the menu “File -> New -> Mico32 Managed Make Project Wizard” and select the “CFIFlashProgrammer” in the option “Select Project Templates.”

Step 2: Build the project successfully and the elf file is generated.
Step 3: Two files (flashprog.bin and flashprog.txt) should be added to the project root directory.

**flashprog.bin**: The image you want to write to the flash, Binary data must be contained in a binary file named flashprog.bin and binary datafile containing the data for programming must be a multiple of 4-bytes.

**flashprog.txt**: A text file must contain two lines, the first line must contain the absolute flash-address where the data needs to be programmed and the second line must contain the size of image to program, specified as bytes. Flash address where the binary data want to be programmed is expected to be aligned on a word (4-bytes) boundary.

For example:

According to the Figure3, assume to program the OS image (jsp.bin) to the flash address 0x04080000, file size is 31764(bytes).

1. Copy the jsp.bin to the “CFIFlashProgrammer” project root directory and rename the file name “jsp.bin” to “flashprog.bin”

2. Create one text file named “flashprog.txt” in project root directory, the content of the text file as following:
   
   0x40800000
   31764

March, 2008
Step 4: Select the "Run"->"Run ..." in the LatticeMico32 system GUI. A window named "Run" will show and double click the "mico32 hardware". set the "Project:" and "C/C++ Application" in the tab "Main". Click the button "Run", the console terminal shows you the flash write is done.

Following is the captured picture

Step 5: Delete the flashprog.bin, copy the boot.bin to the “CFIFlashProgrammer” project root directory and rename the boot.bin to flashprog.bin, then open the flashprog.txt and modify the two lines, the first line was modified to 0x04000000 according to the Figure 3, the second line was updated to 768 (bytes) according to the size of the boot.bin.

Step 6. Go to Step 4. Repeat the step 5, step 6 and step 4 can write many images to the target flash many times as you like.

6.5  Link file for Store and Run the image in the internal memory

/*
 */

OUTPUT_FORMAT("elf32-lm32")
ENTRY(reset)

/*
 * This section defines memory attributes (name, origin, length) for the platform
 */
MEMORY
{
  ebr : ORIGIN = 0x00100000, LENGTH = 32768
  sram : ORIGIN = 0x00200000, LENGTH = 1048576
}

SECTIONS
{

  /* code */
  .boot : { *(.boot) } > ebr
  .text :
  {
    . = ALIGN(4);
    _ftext = .;
    *(.text .stub .text.* .gnu.linkonce.t.*)
    *(.gnu.warning)
    KEEP (*(.init))
    KEEP (*(.fini))

    /* Exception handlers */
    *(.eh_frame_hdr)
    KEEP (*(.eh_frame))
    *(.gcc_except_table)

    /* Constructors and destructors */
    KEEP (*crtbegin*.o(.ctors))
    KEEP (*EXCLUDE_FILE (*.crtend*.o ) .ctors))
    KEEP (*SORT(.ctors.*))
    KEEP (*(.ctors))
    KEEP (*crtbegin*.o(.dtors))
    KEEP (*EXCLUDE_FILE (*crtend*.o ) .dtors))
    KEEP (*SORT(.dtors.*))
    KEEP (*(.dtors))
    KEEP (*(.jcr))
    _etext = .;
  } > ebr =0

  /* read-only data */
  .rodata :
  {
    . = ALIGN(4);
    _frodata = .;
    _frodata_rom = LOADADDR(.rodata);
    *(.rodata .rodata.* .gnu.linkonce.r.*)
*(.rodata)
  _erodata = ;;
} > ebr

/* read/write data */
.data:
{
  . = ALIGN(4);
  _data = ;;
  *(.data .data.* .gnu.linkonce.d.*)
  *(.data1)
  SORT(CONSTRUCTORS)
  _gp = ALIGN(16) + 0x7ff0;
  *(.sdata .sdata.* .gnu.linkonce.s.*)
  _edata = ;;
} > ebr

/* bss */
.bss:
{
  . = ALIGN(4);
  _fbss = ;;
  *(.dynsbss)
  *(.sbss .sbss.* .gnu.linkonce.sb.*)
  *(.scommon)
  *(.dynbss)
  *(.bss .bss.* .gnu.linkonce.b.*)
  *(COMMON)
  . = ALIGN(4);
  _ebss = ;;
  _end = ;;
  PROVIDE (end = .);
} > sram

/* first location in stack is highest address in ram */
PROVIDE(_fstack = ORIGIN(sram) + LENGTH(sram) - 4);

/* stabs debugging sections. */
.stab  0 : { *(.stab) }
.stabstr 0 : { *(.stabstr) }
.stab.excl 0 : { *(.stab.excl) }
.stab.exclstr 0 : { *(.stab.exclstr) }
.stab.index 0 : { *(.stab.index) }
.stab.indexstr 0 : { *(.stab.indexstr) }
.comment 0 : { *(.comment) }

/* DWARF debug sections. */
Symbols in the DWARF debugging sections are relative to the beginning of the section so we begin them at 0. */
/* DWARF 1 */
.debug 0 : { *(.debug) }
.line 0 : { *(.line) }
/* GNU DWARF 1 extensions */
.debug_srcinfo 0 : { *(.debug_srcinfo) }
.debug_sfnames 0 : { *(.debug_sfnames) }
/* DWARF 1.1 and DWARF 2 */
.debug_aranges 0 : { *(.debug_aranges) }
.debug_pubnames 0 : { *(.debug_pubnames) }
/* DWARF 2 */
.debug_info 0 : { *(.debug_info .gnu.linkonce.wi.*) }
.debug_abbrev 0 : { *(.debug_abbrev) }
.debug_line 0 : { *(.debug_line) }
.debug_frame 0 : { *(.debug_frame) }
.debug_str 0 : { *(.debug_str) }
.debug_loc 0 : { *(.debug_loc) }
.debug_macinfo 0 : { *(.debug_macinfo) }
/* SGI/MIPS DWARF 2 extensions */
.debug_weaknames 0 : { *(.debug_weaknames) }
.debug_funcnames 0 : { *(.debug_funcnames) }
.debug_typenames 0 : { *(.debug_typenames) }
.debug_varnames 0 : { *(.debug_varnames) }
}

6.6 Link file for store the image in flash and run in the sram
/*
*/

OUTPUT_FORMAT("elf32-lm32")
ENTRY(reset)

/ *
* This section defines memory attributes (name, origin, length) for the platform
*/
MEMORY
{
    sram : ORIGIN = 0x02000000, LENGTH = 1048576
}

SECTIONS
{
    /* code */
    .boot : { *(.boot) } > sram
    .text :
    {
        . = ALIGN(4);
        _ftext = .;
        *(.text.stub .text.* .gnu.linkonce.t.*)
        *(.gnu.warning)
KEEP (*(.init))
KEEP (*(.fini))

/* Exception handlers */
*(.eh_frame hdr)
KEEP (*(.eh_frame))
*(.gcc_except_table)

/* Constructors and destructors */
KEEP (*crtbegin*.o(.ctors))
KEEP *(EXCLUDE_FILE (*crtend*.o ) .ctors))
KEEP *(SORT(.ctors.*))
KEEP (*(.ctors))
KEEP (*crtbegin*.o(.dtors))
KEEP *(EXCLUDE_FILE (*crtend*.o ) .dtors))
KEEP *(SORT(.dtors.*))
KEEP *(.dtors))
KEEP *(.jcr))
__etext = .;
} > sram =0

/* read-only data */
.rodata :
{
  . = ALIGN(4);
  _frodata = .;
  _frodata_rom = LOADADDR(.rodata);
  *(.rodata .rodata.* .gnu.linkonce.r.*)
  *(.rodata1)
  _erodata = .;
} > sram

/* read/write data */
.data :
{
  . = ALIGN(4);
  _pdata = .;
  *(.data .data.* .gnu.linkonce.d.*)
  *(.data1)
  SORT(CONSTRUCTORS)
  _gp = ALIGN(16) + 0x7ff0;
  *(.sdata .sdata.* .gnu.linkonce.s.*)
  _edata = .;
} > sram

/* bss */
.bss :
{
  . = ALIGN(4);
  _fbss = .;
/* .dynsbss */
/* .sbss .sbss.* .gnu.linkonce.sb.* */
/* .scommon */
/* .dynbss */
/* .bss .bss.* .gnu.linkonce.b.* */
*(COMMON)
   .= ALIGN(4);
   _ebss = .;
   _end = .;
   PROVIDE (end = .);
} > sram

/* first location in stack is highest address in ram */
PROVIDE(_fstack = ORIGIN(sram) + LENGTH(sram) - 4);

/* stabs debugging sections. */
.stab 0 : { *(.stab) }
.stabstr 0 : { *(.stabstr) }
.stab.excl 0 : { *(.stab.excl) }
.stab.exclstr 0 : { *(.stab.exclstr) }
.stab.index 0 : { *(.stab.index) }
.stab.indexstr 0 : { *(.stab.indexstr) }
.comment 0 : { *(.comment) }

/* DWARF debug sections. */
Symbols in the DWARF debugging sections are relative to the beginning
of the section so we begin them at 0. */
/* DWARF 1 */
.debug 0 : { *(.debug) }
.line 0 : { *(.line) }
/* GNU DWARF 1 extensions */
.debug_srcinfo 0 : { *(.debug_srcinfo) }
.debug_sfnames 0 : { *(.debug_sfnames) }
/* DWARF 1.1 and DWARF 2 */
.debug_aranges 0 : { *(.debug_aranges) }
.debug_pubnames 0 : { *(.debug_pubnames) }
/* DWARF 2 */
.debug_info 0 : { *(.debug_info .gnu.linkonce.wi.*) }
.debug_abbrev 0 : { *(.debug_abbrev) }
.debug_line 0 : { *(.debug_line) }
.debug_frame 0 : { *(.debug_frame) }
.debug_str 0 : { *(.debug_str) }
.debug_loc 0 : { *(.debug_loc) }
.debug_macinfo 0 : { *(.debug_macinfo) }
/* SGI/MIPS DWARF 2 extensions */
.debug_weaknames 0 : { *(.debug_weaknames) }
.debug_funcnames 0 : { *(.debug_funcnames) }
.debug_typenames 0 : { *(.debug_typenames) }
.debug_varnames 0 : { *(.debug_varnames) }
}
Group the diagram.

The diagram is not grouped. Group the diagram.