

# MAD Utils User Guide

---

## Multicore Application Deployment (MAD) Utilities User's Guide

Last updated: 07/28/2014

---

### Contents

---

#### Preface

#### Overview

- Acronyms and definitions
- Motivation for MAD infrastructure
- MAD infrastructure components
  - Build time utilities
  - Run time utilities
- Usage modes
  - MAD flow in Prelinker bypass mode
    - Flow summary
  - MAD flow in Prelinker mode
    - Flow summary

#### Getting started

- Package overview
  - MAD UTILS overview
  - MAP tool configuration
  - Deployment configuration file (Prelinker bypass mode)
  - Deployment configuration file (Prelinker mode)
  - MAP tool invocation
- Demo walkthrough
  - Prelinker bypass mode demo
  - Prelinker mode demo
- Software overview
  - MAD Loader overview
    - Code organization
    - Build instructions
    - MAD Loader APIs
- Debugging application on target
  - Prelinker mode:
  - Prelinker bypass mode:
  - Loading and running MAD linked image using CCS:

#### Useful references

## Preface

This document describes the usage of Multicore Application Deployment (MAD) utilities.

## Overview

### Acronyms and definitions

---

The following acronyms are used throughout this document.

Acronym	Description
API	Application Programming Interface
DSO	Dynamic Shared Object
DSBT	Data Segment Base Table

DP	Data Page register
ELF	Executable and Linkable Format
IBL	Intermediate Boot Loader
JSON	Java Script Object Notation
MAD	Multi-core Application Deployment
MAP	Multiple Application Pre-Linker
NML	No Man's Land (Reserved virtual address space)
SP	Stack Pointer register
ROMFS	ROM File System
XIP	eXecute In Place
EEPROM	Electrically Erasable Programmable Read-Only Memory
OFD	Object File Dump
I2C	Inter-Integrated Circuit

## Motivation for MAD infrastructure

---

1. Need to deploy multiple applications on multiple cores.
2. Need to conserve memory by sharing common code.
3. Need to deploy an application dynamically on a core.(Feature not supported currently)

## MAD infrastructure components

---

MAD infrastructure provides a set of utilities to help achieve the above mentioned needs.

There are 5 major utilities used by the MAD infrastructure grouped into the following 2 categories

### Build time utilities

- Static Linker: For linking the applications and dependent dynamic shared objects (DSO).
- Prelink Tool: For binding segments in an ELF file to virtual addresses.
- MAP tool: Multi-core Application Prelinker (MAP) tool to assign virtual address to segments for multicore applications. Following is brief of the MAP tool functionality.

User specifies the desired memory partition for the device and high level instructions for segment placements to the MAP tool. Based on this information MAP tool determines the runtime virtual/physical address for each ELF segment for each application. It then invokes the prelinker to do the storage allocation(address binding) for all the applications and the dependent DSO(s). The MAP tool also generates a set of activation records for loading an application on a specific core. The activation records are instructions to the run-time loader to do the following:

- Setup virtual memory mapping and memory protection/permission attributes of partitions
- Copy and initialize loadable segment at their run address

The prelinked applications, DSO(s) and the activations records are then packed into a ROM file system image to be downloaded to the target.

### Run time utilities

#### Initial Load tool (IBL)

Intermediate Boot Loader (IBL) provides the functionality of downloading the ROM file system image to the device's shared external memory (DDR). The configuration parameters for IBL are programmed into the I2C EEPROM of the target platform.

#### Run Load tool (MAD Loader)

MAD Loader utility provides the functionality of starting an application on a given core. It does the following to start an application on a core.

- Configures the virtual memory map for the core.
- Configures the memory attributes and permissions for each memory partition
- Copies segment from the load address to the run address.

- Initialize the execution environment for the application.
- Execute the pre-initialization functions of the application.
- Execute the initialization functions of the dependent libraries and the application.
- Start the application at its entry point.

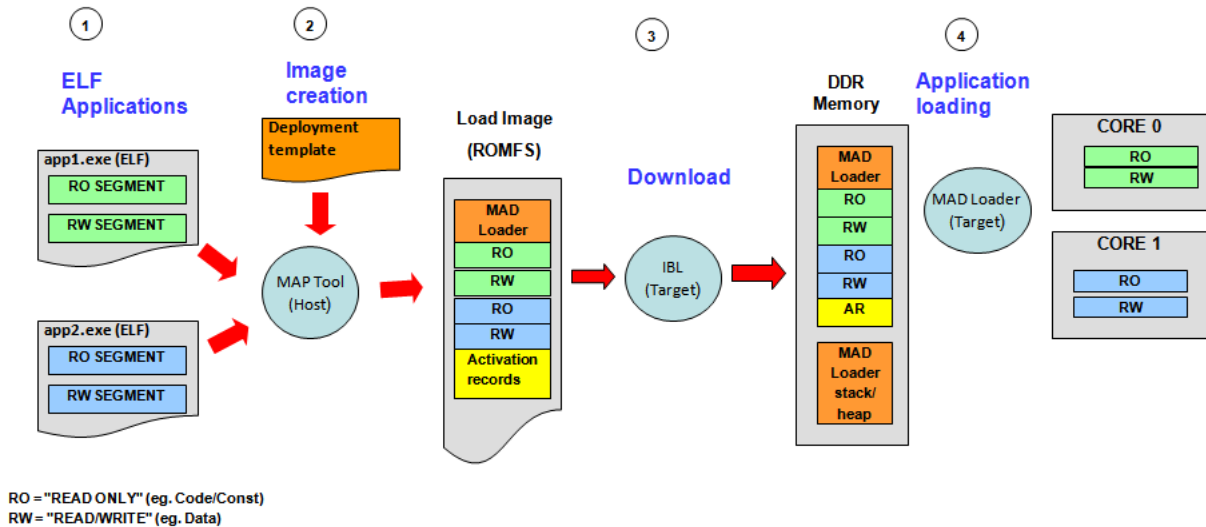
## Usage modes

MAD utilities provide 2 modes of usage

1. **Prelinker bypass mode:** In this mode of operation the MAP tool does not do address assignment for the application segments and the prelinker is not invoked. This mode is suitable for use cases where the application developer has taken care of multicore address assignment for the applications and just needs an utility to load and run the applications on specified cores.
2. **Prelinker mode:** In this mode of operation the MAP tool does the address allocation for the application segments and invokes the prelinker. This mode is suitable for use cases where the application developer wants the MAP tool to take care of address assignment to enable sharing of common code among multicore applications.

## MAD flow in Prelinker bypass mode

The following figure illustrates the MAD Flow in Prelink bypass mode



### Flow summary

#### Image Preparation

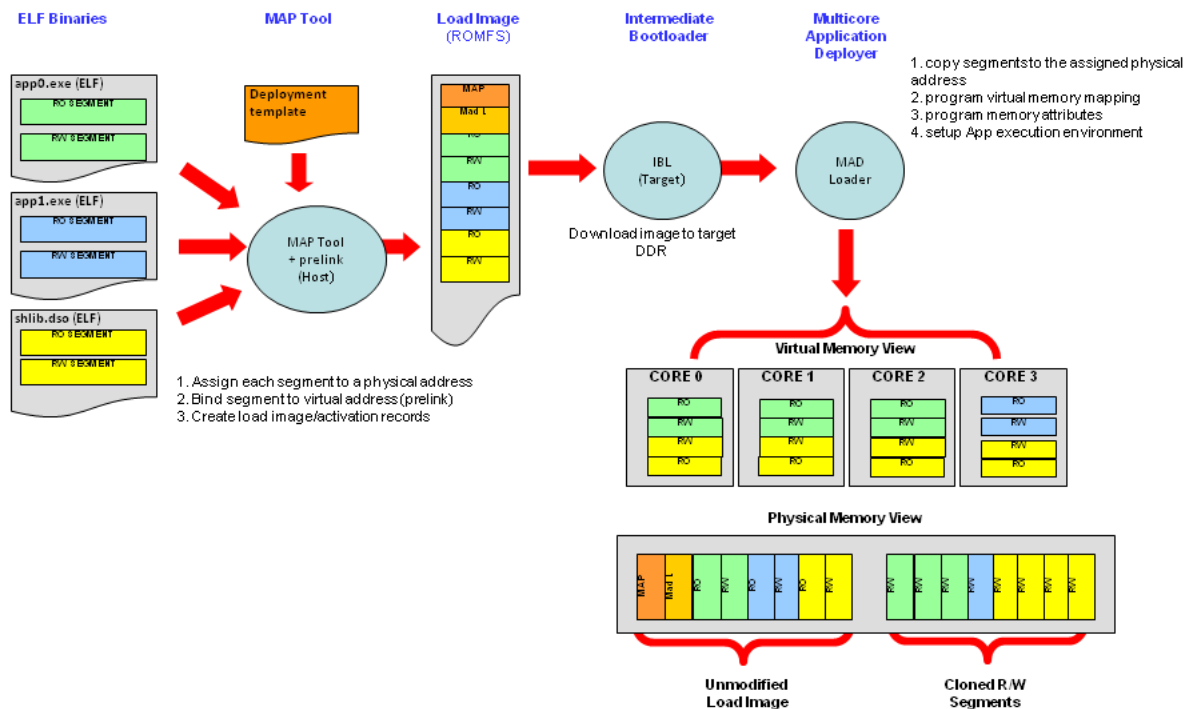
- Statically link the applications to their run address.
- Create a deployment configuration file for the Map tool identifying the application to be loaded on each core.
- Run the MAP tool with the deployment configuration file as an input.
- MAP tool creates a load image (in ROMFS format) containing activation records for each application.

#### Application deployment

- At boot, device will run ROM bootloader
- ROM bootloader will load and run IBL that will be on the board (eg. i2c eeprom)
- IBL will download MAD image from a tftp server (MAD image can also reside onboard on NOR/NAND flash) to DDR
- IBL is configured with an entry point into the execution.
  - In a non-MAD case, this would be entry point of the application downloaded
  - In MAD case, IBL will be configured to jump to the entry point of MAD Loader
- MAD loader will then
  - Parse ROMFS image
  - Load application segments to their run address and start execution of the application on each configured core.

## MAD flow in Prelinker mode

The following figure illustrates the MAD Flow in prelinker mode



**Flow summary**

**Image Preparation**

- Identify common code among applications.
- Link common code as position independent shared objects (DSO).
- Link the applications.
- The above steps create the set of applications/DSO(s) that will be run on the device.
- Identify which applications can run on each core.
- Identify the memory partitions for the device by envisioning the usage.

Example memory partition for C6670 <syntaxhighlight lang="text">

```
DDR:
32 MB per core 0-3
128 MB ro image
16 MB shared r/w
MSMC RAM:
512 KB per core
no ro image
no shared r/w
L2:
Max Cache
L1:
All Cache
```

</syntaxhighlight>

- Create a deployment configuration file for the Map tool with the above information
- Run the MAP tool with the deployment configuration file as an input.
- Map tool generates a pre-link command file containing segment to virtual address binding instructions for the Prelink tool.
- The Prelink tool reads the ELF files in combination with the prelink command file from the Map tool and prelinks all input applications (binding segments to virtual addresses, processing dynamic relocations, etc.) and produces a prelinked output file for every EXE and DSO that was prelinked.
- Using the prelinked output files and the information of the allocation of the segments into the physical address space, the MAP tool creates a load image (in ROMFS format) containing activation records for each application.

**Application deployment**

- At boot, device will run ROM bootloader
- ROM bootloader will load and run IBL that will be on the board (eg. i2c eeprom)
- IBL will download MAD image from a tftp server (MAD image can also reside onboard on NOR/NAND flash) to DDR
- IBL is configured with an entry point into the execution.
  - In a non-MAD case, this would be entry point of the application downloaded

- In MAD case, IBL will be configured to jump to the entry point of MAD Loader
- MAD loader will then
  - Parse ROMFS image
  - Load application segments to their run address and start execution of the application on each configured core.

## Getting started

### Package overview

The following tool packages are required for supporting multicore application deployment

Package	Description
Code Generation Tools	Provides tools for compiling and linking the applications. Also provides the prelinker used by the MAP tool.
MAD utils	Provides the MAP tool, MAD loader
IBL	Provides the Intermediate Boot Loader

**Table 1: Package overview**

### MAD UTILS overview

MAD utils is provided as a source distribution. Following is the directory structure of mad-utils.

directory	description
./mad-loader	Source code for the MAD loader
./map-tool	Source code for MAP tool

**Table 2: MAD UTILS directory structure**

The README.txt files within the respective directories give details about the code organization and build procedures.

### MAP tool configuration

#### MAP tool configuration file

The input to MAP tool is a configuration file in JSON format. The configuration file has the following objects:

1. **deploymentCfgFile** : specifies the deployment configuration file.
2. **LoadImageName** : specifies the name of the load image file to be generated. The load image file (in ROMFS format) will be placed in the ./image directory.
3. **prelinkExe**: specifies the name of the prelinker executable. The path to prelinker executable should be set in the set in the execution environment.
4. **ofdTool**: specifies the name of the OFD tool executable. The path to OFD tool executable should be set in the set in the execution environment. The OFD tool is a part of the code generation tools package.
5. **malApp**: specifies the file name of the MAD loader application
6. **nmlLoader**: specifies the file name of the NML loader. NML loader is a sub-component of the MAD loader.

Sample configuration file for the MAP tool is shown below.

```
<syntaxhighlight lang="javascript"> { "deploymentCfgFile" : "./config-files/deployment_template_C6678.json", "LoadImageName" : "c6678-le.bin", "prelinkExe" : "prelink6x", "stripExe" : "strip6x", "ofdTool" : "ofd6x", "malApp" : "../mad-loader/bin/C6678/le/mal_app.exe", "nmlLoader" : "../mad-loader/bin/C6678/le/nml.exe" } </syntaxhighlight>
```

### Deployment configuration file (Prelinker bypass mode)

In prelinker bypass mode the deployment configuration file is used to specify the following information

- Address of the Load memory partition
- The applications to be deployed.

The deployment configuration file is in JSON format. Deployment configuration file for "prelinker bypass" mode has the following sections:

1. **deviceName**: This JSON object identifies the target device.
2. **partitions**: This sections identifies the memory partition where the ROMFS image will be loaded in memory. This section has the following configuration parameters :
  1. **name** : Name of the partition. This is used as a partition identifier in MAP tool debug logs
  2. **vaddr** : Virtual address of the partition.
  3. **size** : Size of the memory partition in bytes.

4. **loadPartition** : Specifies if the partition is a Load partition.

Sample deployment configuration in prelinker bypass mode for C6670 device is shown below.

```
<syntaxhighlight lang="javascript"> { "deviceName": "C6670",
"partitions": [ { "name": "load-partition", "vaddr": "0x9e000000", "size": "0x2000000", "loadPartition": true } ],
"applications": [ { { "name": "app1", "fileName": "../mad-loader/examples/app_1/build/app_1.exe", "allowedCores": [0,1,2,3] }, { "name": "app2", "fileName": "../mad-loader/examples/app_2/build/app_2.exe", "allowedCores": [0,1,2,3] } ],
"appDeployment": [ "app1", "app2", "app1", "app2" ] } </syntaxhighlight>
```

## Deployment configuration file (Prelinker mode)

The deployment configuration file is used to specify the following information to the MAP tool.

- The desired memory partitions on each core of the device
- Attributes and access permissions of the partitions
- The applications to be deployed.

The deployment configuration file is in JSON format. Deployment configuration file has the following sections:

1. **deviceName**: This JSON object identifies the target device.
2. **partitions**: This section identifies the memory partitions and its attributes. The user controls the placement of ELF segments into partition, by specifying the segment identifiers (section name). **partitions** is a list of structures. Each structure has the following objects:
  1. **name** : Name of the partition. This is used as a partition identifier in MAP tool debug logs
  2. **vaddr** : Virtual address of the partition. For devices without virtual addressing, this would be the physical address.
  3. **paddr** : Physical address of the partition. This is an ordered list indexed by the device CoreId. The value at a given index specifies the physical address corresponding to the virtual address for that core. For devices without virtual memory addressing, vaddr and paddr would be the same.
  4. **size** : Size of the memory partition in bytes.
  5. **secNamePat** : Section name pattern. This is a regular expression string used to identify ELF segments. MAP tool will place all segments with matching section names into this memory partition.
  6. **cores** : List of applicable cores for this partition.
  7. **permissions** : List of access permissions applicable to the partition. Allowed values are SR (supervisor Read), SW (supervisor Write), SX (supervisor Execute), UR (user Read), UW (User Write), UX (User Execute).
  8. **cacheEnable** : Enable/disable Cache. Allowed values are True, False. This is an optional parameter with a default value of True.
  9. **prefetch** : Enable/disable prefetch. Allowed values are True, False. This is an optional parameter with a default value of False.
  10. **shared** : Specifies if the partition is shared among applications on different cores. Allowed values are True, False.
  11. **loadPartition** : Specifies if the partition is a Load partition. The ROMFS image will be downloaded to this partition. MAP tool will try to make the Execute segments in this partition XIP. Allowed values are True, False. This is an optional parameter with a default value of False. There can/must be only one load partition in the system. MAD loader image is placed XIP in the load partition.
  12. **priority** : Specifies the priority for the virtual memory map. A higher number specifies a higher priority. Priority is used when virtual memory mappings are overlapping. This is an optional parameter with a default value of 0.
3. **applications** : This section specifies the applications to be loaded on the device. **applications** is a list of structures. Each structure has the following objects:
  1. **name** : Specifies a name or alias for the application.
  2. **filename** : File name with full path of the application's ELF executable.
  3. **libPath** : Specifies the path for the shared libraries used by the application.
  4. **allowedCores** : List of cores on which the application can run.
4. **appDeployment**: Specifies the applications to be loaded on each core on the initial boot. This is an ordered list of application names indexed by core-id. If a core has to be booted without an application, then an empty string should be specified.

Sample deployment configuration for C6678 device is shown below.

```
<syntaxhighlight lang="javascript"> {
"deviceName": "C6678",
"partitions": [
{ "name": "ddr-code", "vaddr": "0x9e000000", "paddr": [ "0x81e000000", "0x81e000000", "0x81e000000", "0x81e000000", "0x81e000000", "0x81e000000", "0x81e000000", "0x81e000000", "0x81e000000" ], "size": "0x1000000", "secNamePat": [ "^\\.text", "const", "switch" ], "cores": [0,1,2,3,4,5,6,7], "permissions": [ "UR", "UX", "SR", "SX" ], "cacheEnable": true, "prefetch": true, "priority": 0, "shared": true, "loadPartition": true }, { "name": "ddr-data", "vaddr": "0xD0000000", "paddr": [ "0x800000000", "0x801000000", "0x802000000", "0x803000000", "0x804000000", "0x805000000", "0x806000000", "0x807000000" ], "size": "0x1000000", "secNamePat": [ "stack", "^\\.far$", "args", "neardata", "fardata", "rodata" ], "cores": [0,1,2,3,4,5,6,7], "permissions": [ "UR", "UW", "SR", "SW" ], "cacheEnable": true, "prefetch": true, "priority": 0, "shared": false } ],
"applications": [ { { "name": "app1", "fileName": "../mad-loader/examples/app_1/build/app_1.exe", "libPath": "../mad-loader/examples/shlibs/build", "allowedCores": [0,1,2,3,4,5,6,7] }, { "name": "app2", "fileName": "../mad-loader/examples/app_2/build/app_2.exe", "libPath": "../mad-loader/examples/shlibs/build", "allowedCores": [0,1,2,3,4,5,6,7] } ],
"appDeployment": [ "app1", "app2", "app1", "app2", "app1", "app2", "app1", "app2" ] } </syntaxhighlight>
```

## MAP tool invocation

For prelinker mode MAP tool is invoked as: `<syntaxhighlight lang="javascript"> python maptool.py <maptoolCfg.json> </syntaxhighlight>`

For prelinker bypass mode MAP tool is invoked as: `<syntaxhighlight lang="javascript"> python maptool.py <maptoolCfg.json> bypass-prelink </syntaxhighlight>` where `maptoolCfg.js` on is the input configuration file in JSON format

## Demo walkthrough

---

### Prelinker bypass mode demo

This section will do a walk through of an example MAD flow in prelinker bypass mode on C6670 EVM. The mad-utils package contains a few example applications which will be used here.

#### STEP-1: Image Preparation

- Build the sample applications app\_1, app\_2 in the directory mad-loader/examples/ as static executables.

A build script to build the examples is provided in the directory "mad-loader/examples". To build the examples as static executable invoke the build script as:

```
<syntaxhighlight lang="bash"> ./build_examples_lnx.sh C6670 little static </syntaxhighlight>
```

The above step should create the set of applications that will be run on the device

- Build the MAD loader components:
  - MAD Loader library
  - NML Loader
  - MAD Loader application

The build script provided in "mad-loader" directory can be invoked as follows to build the mad loader executable.

```
<syntaxhighlight lang="bash"> ./build_loader_lnx.sh C6670 </syntaxhighlight>
```

#### STEP-2: Deployment configuration preparation

##### Load partition specification:

```
<syntaxhighlight lang="javascript"> {
```

```

"name"          : "load-partition",
"vaddr"         : "0x9e000000",
"size"          : "0x2000000",
"loadPartition" : true

```

```
</syntaxhighlight>
```

#### STEP-3: Invoke MAP tool

- Create a deployment configuration file for the Map tool with the above information. The sample deployment configuration for this example is available in the mad-utils package as map-tool/config-files

```
/deployment_template_C6670_bypass_prelink.js on.
```

- Create the configuration file for the MAP tool. The sample configuration for this example is available in the mad-utils package as map-tool/config-files

```
/maptoolCfg_C6670_bypass_prelink.js on.
```

- Invoke the MAP tool on a Linux bash shell:

```
<syntaxhighlight lang="bash"> python maptool.py config-files/maptoolCfg_C6670_bypass_prelink.json bypass-prelink </syntaxhighlight>
```

- Check that the output image

c6670-le.bi n has been created in ./images directory.

#### STEP-4: Application deployment on device

The target board should have IBL programmed and configured on the boards I2C EEPROM. Details on programming and configuring IBL are available at <http://linux-c6x.org/wiki/index.php/Bootloaders> (<http://linux-c6x.org/wiki/index.php/Bootloaders>). Following should be the blob configuration parameters to be programmed into IBL configuration:

- startAddress = 0x9e000000.
- branchAddress = 0x9e001040

Place the output image c6670-le.bi n into the root directory of the TFTP server running on the host computer.

Power cycle the target board and wait for ~10 seconds. The device should now have downloaded the ROMFS image and deployed the sample applications on all the cores.

Following is the procedure to verify that the sample application has been successfully deployed on a core.

- connect CCS to a core.
- Load the symbols for the application running on that core.
- Verify that the value of the variable signature is <app name><core id>. E.g. app1core0

### Prelinker mode demo

This section will do a walk through of an example MAD flow in prelinker mode on C6678 EVM. The mad-utils package contains a few example applications and DSO(s) which will be used here.

**STEP-1: Image Preparation**

- Build the sample shared library in the directory mad-loader/examples/shlibs/ as dynamic relocatable object
- Build the sample applications app\_1, app\_2 in the directory mad-loader/examples/ as dynamic relocatable object

A build script to build the examples is provided in the directory "mad-loader/examples". To build the examples as relocatable objects invoke the build script as:

```
<syntaxhighlight lang="bash"> ./build_examples_lnx.sh C6678 little relocatable </syntaxhighlight>
```

The above step should create the set of applications/DSO(s) that will be run on the device

- Build the MAD loader components:
  - MAD Loader library
  - NML Loader
  - MAD Loader application

The build script provided in "mad-loader" directory can be invoked as follows to build the mad loader executable.

```
<syntaxhighlight lang="bash"> ./build_loader_lnx.sh C6678 </syntaxhighlight>
```

**STEP-2: Device memory partitioning & deployment config preparation**

For this example, we will place all the executable segments and all the data segments in DDR memory. For data segments the DDR memory is equally partitioned among the 8 cores. So we need to create 2 memory partitions for the device.

**Partition-1 (for code):**

```
<syntaxhighlight lang="javascript"> {
```

```

"name"      : "ddr-code",
"vaddr"    : "0x9e000000",
"paddr"    : [ "0x81e00000", "0x81e00000", "0x81e00000", "0x81e00000", "0x81e00000", "0x81e00000", "0x81e00000", "0x81e00000" ],
"size"     : "0x1000000",
"secNamePat": [ "^text", "const", "switch" ],
"cores"    : [0,1,2,3,4,5,6,7],
"permissions" : ["UR", "UX", "SR", "SX"],
"cacheEnable" : true,
"prefetch"  : true,
"priority"  : 0,
"shared"    : true,
"loadPartition" : true

```

```
}, </syntaxhighlight>
```

Partition-1 is also marked as the loadPartition, since the downloaded ROMFS image will be placed here. MAP tool will make the executable segments placed in this partition XIP. MAD loader is also placed in this partition.

**Partition-2 (for data):** <syntaxhighlight lang="javascript"> {

```

"name"      : "ddr-data",
"vaddr"    : "0xD0000000",
"paddr"    : [ "0x80000000", "0x80100000", "0x80200000", "0x80300000", "0x80400000", "0x80500000", "0x80600000", "0x80700000" ],
"size"     : "0x1000000",
"secNamePat": [ "stack", "^far$", "args", "neardata", "fardata", "rodata" ],
"cores"    : [0,1,2,3,4,5,6,7],
"permissions" : ["UR", "UW", "SR", "SW"],
"cacheEnable" : true,
"prefetch"  : true,
"priority"  : 0,
"shared"    : false

```

```
} </syntaxhighlight>
```

**STEP-3: Invoke MAP tool**

- Create a deployment configuration file for the Map tool with the above information. The sample deployment configuration for this example is available in the mad-utils package as map-tool/config-files

```
/deployment_template_c6678.js on.
```

- Create the configuration file for the MAP tool. The sample configuration for this example is available in the mad-utils package as map-tool/config-files

```
/maptoolCfg_c6678.js on.
```

- Invoke the MAP tool on a Linux command shell:

```
<syntaxhighlight lang="bash"> python maptool.py config-files/maptoolCfg_c6678.json </syntaxhighlight>
```

- Check that the output image

c6678-le.bin has been created in ./images directory.

**STEP-4: Application deployment on device**

The target board should have IBL programmed and configured on the boards I2C EEPROM. Details on programming and configuring IBL are available at <http://linux-c6x.org/wiki/index.php/Bootloaders> (<http://linux-c6x.org/wiki/index.php/Bootloaders>). Following should be the blob configuration parameters to be programmed into IBL configuration:



- startAddress = 0x9e000000.
- branchAddress = 0x9e001040

Place the output image c6678-le.bi n into the root directory of the TFTP server running on the host computer.

Power cycle the target board and wait for ~10 seconds. The device should now have downloaded the ROMFS image and deployed the sample applications on all the cores.

Following is the procedure to verify that the sample application has been successfully deployed on a core.

- connect CCS to the core. Load the CCS GEL file created by the MAP tool in the ./images directory. This GEL file will have functions to load symbols for the example application.
- Run the GEL function to refresh the symbols for the application running on that core.
- Verify that the value of the variable signature is <app name><core id>. E.g. app1core4

## Software overview

---

### MAD Loader overview

#### Code organization

**./examples:** This folder contains example applications and DSO(s) for testing the MAD flow

**./mal:** This folder contains the source for MAD loader library and the loader application

**./nmlLoader:** This folder contains the source for the no man's land loader(NML). NML is a sub-component of the MAD loader and resides in a reserved virtual address space.

#### Build instructions

**NOTE FOR BUILDING ON WINDOWS ENVIRONMENT:** For building on windows environment GNU utilities like "make" would be required. MINGW-MSYS installation is required for Windows build environment. MINGW-MSYS can be downloaded from <http://sourceforge.net/projects/mingw/files/>

**Build environment Setup:** Before starting the build following environment setup has to be done

- variable C\_DIR should be set to the top directory of the Code Generation tools.
- Code Generation tool binaries should be in the path.

**Linux bash shell:** <syntaxhighlight lang="bash"> export C\_DIR=/opt/TI/TI\_CGT\_C6000\_7.2.4 export PATH=/opt/TI/TI\_CGT\_C6000\_7.2.4/bin:\$PATH </syntaxhighlight>

**MSYS bash shell:** <syntaxhighlight lang="bash"> export C\_DIR="C:/Program Files/Texas Instruments/C6000 Code Generation Tools 7.2.4" export PATH=/c/Program Files/Texas Instruments/C6000 Code Generation Tools 7.2.4/bin:\$PATH </syntaxhighlight>

**Build scripts:** Sample build scripts are provided in the current directory to build MAD loader. Modify the above mentioned environment variables in the build scripts to adjust to existing build environment.

- build\_loader\_inx.sh: Script to build MAD loader in Linux bash shell
- build\_loader\_msys.sh: Script to build MAD loader in MSYS bash shell

#### Build script usage:

- For little endian build the script is invoked as:

```
<syntaxhighlight lang="bash"> ./build_loader_inx.sh device_name </syntaxhighlight>
```

- For big endian build the script is invoked as:

```
<syntaxhighlight lang="bash"> ./build_loader_inx.sh device_name big </syntaxhighlight> where device name can be C6670 or C6678
```

#### Example applications:

The example applications can be built using the sample build script provided in "mad-loader/examples" directory. The above mentioned environment variables have to be modified in the build script. The build script usage is as follows:

```
<syntaxhighlight lang="bash">
```

```
./build_examples_inx.sh C6678|C6670 big|little [static|relocatable] </syntaxhighlight>
```

#### Details about MAD loader build MAD loader library:

The makefile for building the MAD loader library is in the directory "mal/malLib/build"

Following are the steps to build the MAD loader library: <syntaxhighlight lang="bash"> cd mal/malLib/build make DEVICE=<device number: supported device numbers are C6670, C6678> </syntaxhighlight> **MAD loader application:**

The makefile for building the MAD loader library is in the directory "mal/malApp/build"

Since the MAD loader application links the MAD loader library hence the MAD loader library has to be built prior to building MAD loader App

Following are the steps to build the MAD loader library: <syntaxhighlight lang="bash"> cd mal/malApp/build make DEVICE=<device number: supported device numbers are C6670, C6678> </syntaxhighlight>

MAD loader application needs to be XIP in DDR. The linker command file "lnk\_<device number>.cmd" is used to ensure that the MAD loader app is bound to XIP address in DDR. **IMPORTANT NOTE:** MAD loader application needs RW area for stack, heap and global variables. By default the linker command file has been setup to allocate the RW area towards the end of DDR memory. This can be changed by the user according to the target execution environment. User should note that this memory are should not be used by the application being loaded. It should be taken care to avoid using this area in the application/deployment configuration file.

#### NML:

The makefile for building the NML is in the directory "nmlLoader/build"

Following are the steps to build NML: 

```
<syntaxhighlight lang="bash"> cd nmlLoader/build make DEVICE=<device number: supported device numbers are C6670, C6678> </syntaxhighlight>
```

Since NML is XIP in DDR, it has to be ensured that the NML code segments are bound to virtual address which is XIP in DDR. NML is a part of the ROM file system loaded on DDR. if the offset of the NML ELF file changes in the ROM file system, then this address needs to be modified. This can happen if the size of the MAD loader application changes. To get the current offset of NML in ROM file system, do a trial run of the MAP tool, MAP tool will create a file ./tmp/fsOffsets.txt. This file will list the offset of all the files in the filesystem. The linker command file "lnk\_<device number>.cmd" is used to ensure that the NML is bound to XIP address in DDR.

**IMPORTANT NOTE:** NML also needs RW area for stack and global variables. By default the linker command file has been setup to allocate the RW area towards the end of DDR memory. This can be changed by the user according to the target execution environment. User should note that this memory are should not be used by the application being loaded. It should be taken care to avoid using this area in the application/deployment configuration file.

#### MAD Loader APIs

MAD loader library can be linked by an application to provide application deployment services. This section describes the APIs provided by the MAD loader library.

##### Loader APIs:

*int mal\_lib\_init (void \*load\_partition\_addr):* API to initialize the library

*int mal\_lib\_stop\_core (unsigned int coreId):* API to stop a core. Non graceful forced shutdown.

*int mal\_lib\_load\_core (unsigned int coreId, char \*appName):* API to load and run an APP on a given core.

##### File System APIs:

*int mal\_lib\_fopen (const char \*filename):* API to open a file stream

*int mal\_lib\_fclose (int file\_handle):* API to close a file stream

*int mal\_lib\_fsize (const char \*filename, unsigned int \*size):* API to get the file size

*unsigned int mal\_lib\_fread (void \*ptr, unsigned int size, unsigned int count, int file\_handle):* API to read from a file stream

*int mal\_lib\_fseek (int file\_handle, unsigned int offset, int origin):* API to seek to a position in a file

*long mal\_lib\_ftell (int file\_handle):* API to get the current offset in the file

## Debugging application on target

---

#### Prelinker mode:

When MAD tools are used in Prelinker mode the applications are prelinked to memory address as specified in the deployment configuration file. The prelinker does not update the DWARF info in the ELF file while doing relocations. Hence to enable symbolic debugging with the relocated images, the MAP tool generates a CCS GEL file which provides the relocation information of the application to the CCS debug server. This GEL is created in the "./images" directory.

#### Prelinker bypass mode:

In Prelinker bypass mode the application executables are not relocated. Hence target debugging features will function as normal.

#### Loading and running MAD linked image using CCS:

Sometime, it is useful to load and run the output image using CCS for initial testing.

Please follow the steps below to load and run the image using CCS

- Launch CCS and load your configuration file for the EVM
- Connect to core 0
- Open memory browser (View --> Memory) and navigate to 0x9E000000
- Right click on the memory browser and select *Load Memory*; Browse and select the image (\*.bin) generated from the previous step (you may have to change the file-type option to see .bin files); Press Next
- For start address, enter 0x9E000000.
- Be sure of your **Type-size** choice (32 bits in C6678 case). Click finish
- To run the program, open the register browser (View --> Registers) for core 0
- Change the PC (Program Counter) to 0x9E001040

- Run core 0

Note: You won't be able to see output of printf in CCS console, but if your image prints messages in UART you should be able to see the output


## Useful references

**Image processing Demo:** This demo utilizes MAD tools for multicore deployment and can be viewed as an useful example reference. User guide for this is available at [http://processors.wiki.ti.com/index.php/MCSDK\\_Image\\_Processing\\_Demonstration\\_Guide](http://processors.wiki.ti.com/index.php/MCSDK_Image_Processing_Demonstration_Guide) ([http://processors.wiki.ti.com/index.php/MCSDK\\_Image\\_Processing\\_Demonstration\\_Guide](http://processors.wiki.ti.com/index.php/MCSDK_Image_Processing_Demonstration_Guide))

**C66x multicore DSP forum:** For technical support on MAD utils. [http://e2e.ti.com/support/dsp/c6000\\_multi-core\\_dsps/default.aspx](http://e2e.ti.com/support/dsp/c6000_multi-core_dsps/default.aspx) ([http://e2e.ti.com/support/dsp/c6000\\_multi-core\\_dsps/default.aspx](http://e2e.ti.com/support/dsp/c6000_multi-core_dsps/default.aspx))

<pre> {{ 1. switchcategory:MultiCore= ▪ For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum ▪ For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum Please post only comments related to the article <b>MAD Utils User Guide</b> here.         </pre>	<p><b>Keystone=</b></p> <ul style="list-style-type: none"> <li>▪ For technical support on MultiCore devices, please post your questions in the <a href="#">C6000 MultiCore Forum</a></li> <li>▪ For questions related to the BIOS MultiCore SDK (MCSDK), please use the <a href="#">BIOS Forum</a></li> </ul> <p>Please post only comments related to the article <b>MAD Utils User Guide</b> here.</p>	<p><b>C2000=For technical support on the C2000 please post your questions on The C2000 Forum.</b> Please post only comments about the article <b>MAD Utils User Guide</b> here.</p>	<p><b>DaVinci=For technical support on DaVinciplease post your questions on The DaVinci Forum.</b> Please post only comments about the article <b>MAD Utils User Guide</b> here.</p>	<p><b>MSP430=For technical support on MSP430 please post your questions on The MSP430 Forum.</b> Please post only comments about the article <b>MAD Utils User Guide</b> here.</p>	<p><b>OMAP35x=For technical support on OMAP please post your questions on The OMAP Forum.</b> Please post only comments about the article <b>MAD Utils User Guide</b> here.</p>	<p><b>OMAPL1=For technical support on OMAP please post your questions on The OMAP Forum.</b> Please post only comments about the article <b>MAD Utils User Guide</b> here.</p>	<p><b>MAVRK=For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum.</b> Please post only comments about the article <b>MAD Utils User Guide</b> here.</p>
--	---	---	--	--	---	--	---

### Links

 <ul style="list-style-type: none"> <li><a href="#">Amplifiers &amp; Linear Audio</a></li> <li><a href="#">Broadband RF/IF &amp; Digital Radio</a></li> <li><a href="#">Clocks &amp; Timers</a></li> <li><a href="#">Data Converters</a></li> </ul>	<ul style="list-style-type: none"> <li><a href="#">DLP &amp; MEMS High-Reliability Interface</a></li> <li><a href="#">Logic</a></li> <li><a href="#">Power Management</a></li> </ul>	<p><u>Processors</u></p> <ul style="list-style-type: none"> <li>▪ <a href="#">ARM Processors</a></li> <li>▪ <a href="#">Digital Signal Processors (DSP)</a></li> <li>▪ <a href="#">Microcontrollers (MCU)</a></li> <li>▪ <a href="#">OMAP Applications Processors</a></li> </ul>	<p><u>Switches &amp; Multiplexers</u></p> <ul style="list-style-type: none"> <li><a href="#">Temperature Sensors &amp; Control ICs</a></li> <li><a href="#">Wireless Connectivity</a></li> </ul>
---	--	--	--

Retrieved from "[https://processors.wiki.ti.com/index.php?title=MAD\\_Utils\\_User\\_Guide&oldid=182417](https://processors.wiki.ti.com/index.php?title=MAD_Utils_User_Guide&oldid=182417)"

This page was last edited on 28 July 2014, at 05:00.

Content is available under [Creative Commons Attribution-ShareAlike](#) unless otherwise noted.