


Embedded Linux

Home (<https://numato.com/help>) / Knowledge Base (<https://numato.com/kb/>) / Saturn Spartan 6 FPGA Module (<https://numato.com/kb-category/saturn-spartan-6-fpga-module/>)





 Have a question? Enter a search term

SEARCH

Popular Search: USB GPIO (), USB Relay (), FPGA ()



Saturn, Microblaze and Linux – How to Run Linux on Saturn Spartan 6 FPGA Module – Part IV

 1452 views  March 9, 2016  admin  7



(https://numato.com/help/wp-content/uploads/2016/03/microblaze_Linux.png)

- **Part I** (<http://docs.numato.com/knowledge/saturn-microblaze-and-linux-how-run-linux-saturn-spartan-6-fpga-module-part-i/>)
- **Part II** (<https://numato.com/kb/saturn-microblaze-and-linux-how-run-linux-saturn-spartan-6-fpga-module-part-ii/>)
- **Part III** (<http://docs.numato.com/knowledge/saturn-microblaze-and-linux-how-run-linux-saturn-spartan-6-fpga-module-part-iii/>)
- **Part IV**

In previous articles in this series, we saw how to create a Microblaze based embedded platform for Saturn Spartan 6 FPGA module (<https://numato.com/saturn-spartan-6-fpga-development-board-with-ddr-sdram>), how to build Linux kernel and boot Linux on Saturn using Xilinx Platform cable USB and XMD. Though this is the easier method, there are a couple of drawbacks associated to it. You will need a Xilinx Platform Cable USB which is slightly expensive and the images are programmed SRAM/DDR instead of SPI flash. It is possible to program the onboard SPI flash with the images which would retain contents even after power cycle and this can be done without buying expensive JTAG cables.

Before we can proceed, we will need to make a minor change to our XPS project and rebuild it. XPS always generate a new project with startup clock set to JTAG clock. This is great when we use Xilinx Platform Cable USB to download the bit file directly to FPGA SRAM. But since in this part of the article we are going to finally download the image in to SPI flash and let the FPGA load the configuration on it's own from SPI flash, we will need to change the startup clock to CCLK. To do this, open the XPS project and go to the "Project" tab next to "IP Catalog" tab and locate Bitgen options "bitgen.ut". Double click to open it and change the line "-g StartUpClk:JTAGCLK" to "-g StartUpClk:CCLK". Save bitgen.ut and rebuild the entire project by clicking "Generate Bitstream" button on the left side toolbar. This step is very critical to generate a bootable image and please don't skip it.

To program the onboard SPI flash with, we will need to make a single binary image that will contain the following individual images.

- FPGA bit file for Microblaze
- Linux kernel
- A bootloader



A bootloader is necessary here to copy the Linux kernel from SPI flash to DDR and execute it. the primary requirements for the bootloader is that it should be capable of reading data from the SPI flash and should fit within the BRAM cache size.

To build a bootloader, create a new SDK workspace and create a new application project. Let's call this project "saturn_v3_bootloader". You can find more information about creating a SDK project in the article [Creating Xilinx EDK test project for Saturn – Your first Microblaze processor based embedded design](https://numato.com/kb/creating-xilinx-edk-test-project-saturn-your-first-microblaze-processor-based-embedded-design/) (<https://numato.com/kb/creating-xilinx-edk-test-project-saturn-your-first-microblaze-processor-based-embedded-design/>). When selecting the application template, select "Hello World". Once the project is created, open the "SaturnV3Bootloader" project in the "Project Explorer" on the left. Double click on the linker script lscript.ld and make sure that all sections are mapped to BRAM (should look something like microblaze_0_i_bram_ctrl_microblaze_0_d_bram_ctrl). If not, map all sections to BRAM manually and save the linker script. After saving linker script, build the project and make sure that no errors are reported.

Open HelloWorld.c in the project and replace the entire code with the code below. Please note, the code below is a bare-bones bootloader designed to be simpler than being robust.



```

#include <stdio.h>
#include "platform.h"
#include "xspi.h"
#include "xparameters.h"
#include "xil_cache.h"

void print(char *str);

//Set the offset and size of image in SPI flash
#define FLASH_IMAGE_START_ADDRESS 0x500000
#define FLASH_IMAGE_SIZE 0x600000

//Set the address where image will be loaded. This will usually point to
//DDR or SRAM depending on the board architecture. Remember to build
//your application/Linux kernel with this address as base address
#define IMAGE_LOAD_ADDRESS XPAR_LPDDR_S0_AXI_BASEADDR

//Define ID of the SPI peripheral that is connected to the SPI flash
#define SPI_DEVICE_ID XPAR_AXI_SPI_0_DEVICE_ID

void (*imageEntry)();
XSpi Spi;

//This function reads a byte from SPI peripheral
u8 spiReadData()
{
    while(!(XSpi_ReadReg(Spi.BaseAddr, XSP_SR_OFFSET) & 0x02));
    return XSpi_ReadReg(Spi.BaseAddr, XSP_DRR_OFFSET);
}

//This function writes one byte to the SPI peripheral
void spiWriteData(u8 data)
{
    while(XSpi_GetStatusReg(&Spi) & 0x08);
    XSpi_WriteReg(Spi.BaseAddr, XSP_DTR_OFFSET, data);
}

//This function loads the image to the destination (DDR/SRAM) and executes it
int loadAppImage()
{
    XSpi_Config *cfgPtr;
    u8 recBuffer[4];
    u32 i = 0, index = 0, ddrPtr = 0;

    print("Initializing Numato Saturn V3 SPI Image Loader...\n\r");
    print("*** http://numato.com ***\n\r");
    print("\n\r");

    //Lookup SPI peripheral configuration details
    cfgPtr = XSpi_LookupConfig(SPI_DEVICE_ID);
    if (cfgPtr == NULL)
    {
        return XST_DEVICE_NOT_FOUND;
    }
}

```



```

if(XSpi_CfgInitialize(&Spi, cfgPtr, cfgPtr->BaseAddress) != XST_SUCCESS)
{
    return XST_FAILURE;
}

//Beyond this point we will use only low level APIs in favor of smaller
//and simpler code.

//Set up SPI controller. Master, manual slave select. The SPI peripheral
//is configured with no FIFO
XSpi_SetControlReg(&Spi, 0x86);

//Disable interrupts
XSpi_IntrGlobalDisable(&Spi);

//Cycle CS to reset the flash to known state
XSpi_WriteReg(Spi.BaseAddr, XSP_SSR_OFFSET, 0x00);
XSpi_WriteReg(Spi.BaseAddr, XSP_SSR_OFFSET, 0x01);
XSpi_WriteReg(Spi.BaseAddr, XSP_SSR_OFFSET, 0x00);

//Write command 0x0b (fast read) to SPI flash and do a dummy read
spiWriteData(0x0b);
spiReadData();

//Send the address from where the image needs to be loaded.
//Dummy read after every write as usual
spiWriteData((FLASH_IMAGE_START_ADDRESS >> 16) & 0xff);
spiReadData();
spiWriteData((FLASH_IMAGE_START_ADDRESS >> 8) & 0xff);
spiReadData();
spiWriteData((FLASH_IMAGE_START_ADDRESS) & 0xff);
spiReadData();

//A dummy write/read as per W25Q128FV datasheet
spiWriteData(0x00);
spiReadData();

print("Loading application image...\n\r");

for(i=0; i<=FLASH_IMAGE_SIZE; i++)
{
    //Do a dummy write
    spiWriteData(0x00);

    //Read data back
    recBuffer[index] = spiReadData();
    index++;

    //Write the data to DDR/SRAM four bytes at a time
    if(index >= 4)
    {
        *((u32*)(ddrPtr + IMAGE_LOAD_ADDRESS)) = *((u32*)&recBuffer);
        ddrPtr += 4;
        index = 0;
    }
}

```



```

    }
}

print("Executing application image...\n\r");
//Invalidate instruction cache to clean up all existing entries
Xil_ICacheInvalidate();
//Execute the loaded image
imageEntry = (void (*)(void))IMAGE_LOAD_ADDRESS;
(*imageEntry)();

//We shouldn't be here
return 0;
}

int main()
{
    init_platform();
    loadAppImage();
    return 0;
}

```

Rebuild the project with newly added code and make sure that no errors reported. If any errors pop up, please fix them before continuing.

The above code will load the kernel image from SPI flash memory and write to DDR and jump to kernel entry point. This code assumes that the kernel image is sitting at a specific address in the SPI flash (0x500000 in this case) and has a specific size (0x600000) in this case. 0x600000 is little bigger than the actual kernel image but that shouldn't cause any problems. The area starting from 0x000000 to 0x4FFFFFF is reserved for FPGA configuration data (enough to hold configuration data for devices as big as XC6SLX150). So in essence, we are targeting the following memory map for the SPI flash binary image we are going to create.

1. 0x000000 – 0x4FFFFFF -> FPGA configuration data
2. 0x500000 – 0xAFFFFFF -> Kernel image
3. 0xB00000 upwards -> User data (not used in our case)

At this point, we have the bit file for FPGA configuration, Linux kernel image and bootloader image we just built. Before we can proceed with packing these images in to a single binary file, there is one crucial step we need to do. You may remember that fact the the kernel image we built (simpleImage.saturn_v3) is in ELF format. This is helpful when downloading kernel using XMD but the metadata (in addition to executable code) is going to be a problem since our bootloader does not understand ELF format. One option is to add ELF loader code to bootloader but it is going to be time consuming and simply an overkill for such a simple implementation like this. Another option would be to strip the metadata and generate a pure executable binary file. This can be easily done with objcopy (<https://sourceware.org/binutils/docs/binutils/objcopy.html>) tool (objcopy is part of gnu binutils). But the tricky thing

here is that you can not use any objcopy (eg: objcopy built for host machine). You will need to use objcopy that is built for Microblaze. Fortunately, buildroot already built cross objcopy that runs on host machine but can work on Microblaze executables. This happened when we built the Linux kernel earlier (buildroot builds all cross tools needed for building the kernel). All that we need to do is to find where these cross compiled tools are placed by buildroot. With the specific buildroot version we used, the objcopy executable we need was placed in the directory "output/host /usr/microblazeel-buildroot-linux-gnu/bin" under the buildroot root directory. Switch to output/images directory under buildroot root directory where Linux kernel image is available in ELF format and execute the following command.

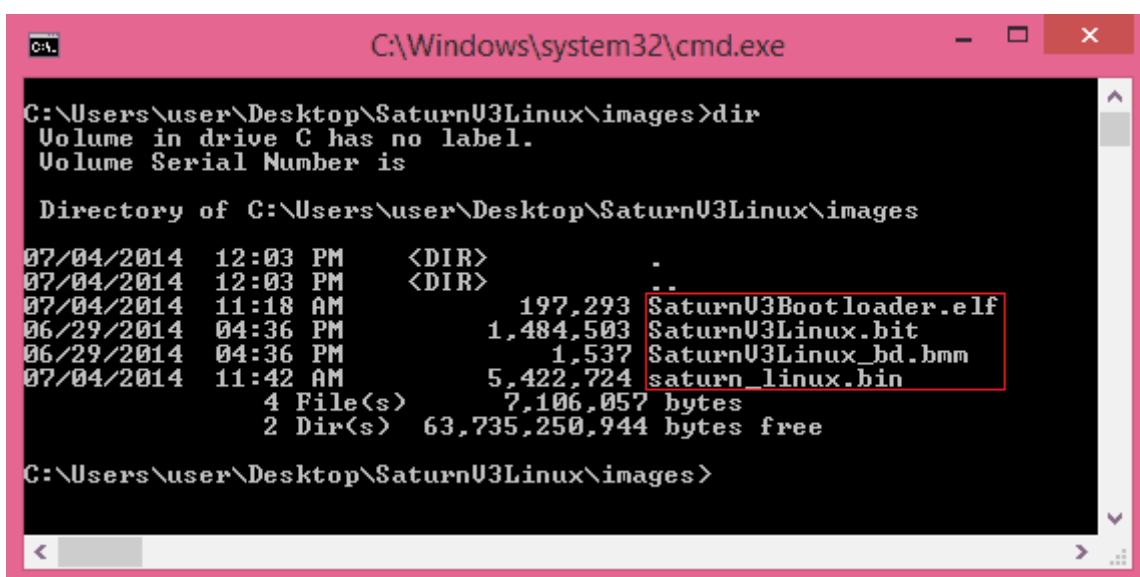
```
cmd>../host/usr/microblazeel-buildroot-linux-gnu/bin/objcopy -O binary simpleImage.saturn
```

If the command was successful, you will see a new file created with the name saturn_linux.bin. Copy this file over to the Windows machine.

On the Windows machine, place the following files in the same directory.

1. The Linux image in binary format (saturn_linux.bin we just created)
2. The bit file (.bit) for the Microblaze system and the Block ram Memory Map file (.bmm) (Both files can be found in the folder SDK\SDK_Export\hw under the XPS project folder)
3. The bootloader executable (This file can be found in the directory SaturnV3Bootloader\Debug under the SDK bootloader project folder and has .elf extension)

In my case, I have the following files in my directory as shown in the image. Your file names may be different depending on the project names etc..



(<https://numato.com/help/wp-content/uploads/2016/03/allimages1.png>)



With all necessary images available now, let us move forward with creating the final

binary image. We will do the following steps to create a single binary image.

1. Merge the FPGA configuration (.bit file) and the Bootloader executable image (.elf). This is because the contents of the executable needs to go in to the FPGA BRAM when FPGA initialize after power up.
2. Concatenate the combined FPGA configuration file (generated in previous step) with Linux kernel image.

To merge Bootloader executable image with bit file, run the following command.

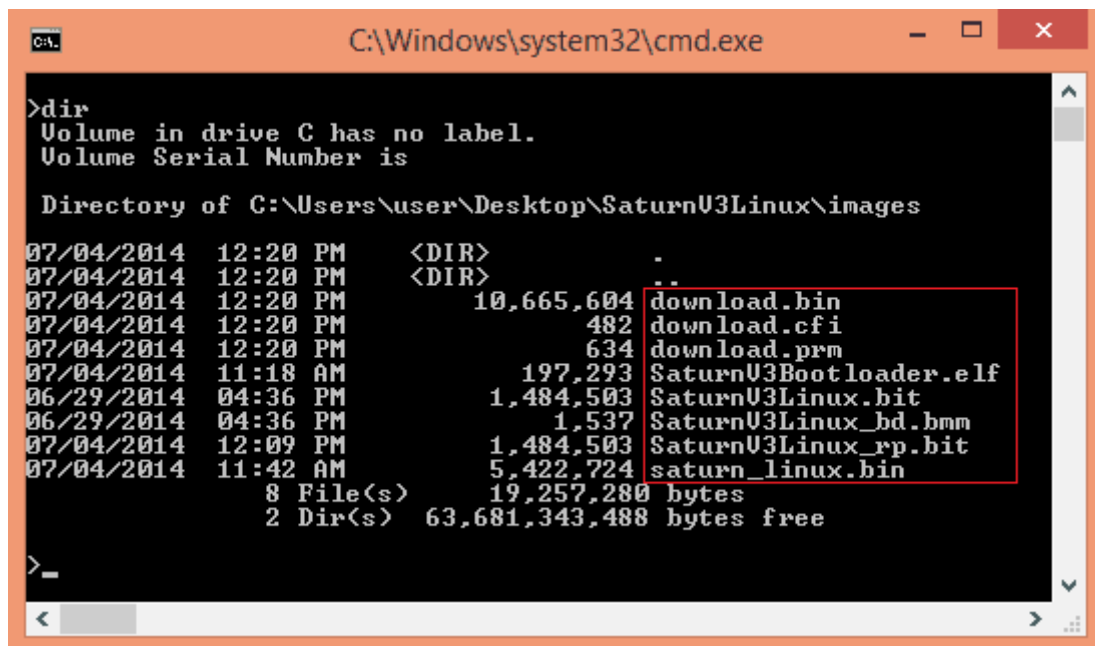
```
cmd>c:\Xilinx\14.6\ISE_DS\settings64.bat
cmd>data2mem -bm SaturnV3Linux_bd.bmm -bd SaturnV3Bootloader.elf -bt SaturnV3Linux.bit -v
```

If the command ran successfully, you will see a new file created in the directory with name "SaturnV3Linux_rp.bit". This file now has both the FPGA configuration stream and Bootloader code.

Now we can concatenate the FPGA configuration file we just generated (SaturnV3Linux_rp.bit) with Linux kernel binary image (saturn_linux.bin) to create a single binary image that can be downloaded to the SPI flash. Run the following command to generate single binary image.

```
cmd>promgen -w -p bin -c FF -o download -s 16384 -u 0 SaturnV3Linux_rp.bit -data_file up
```

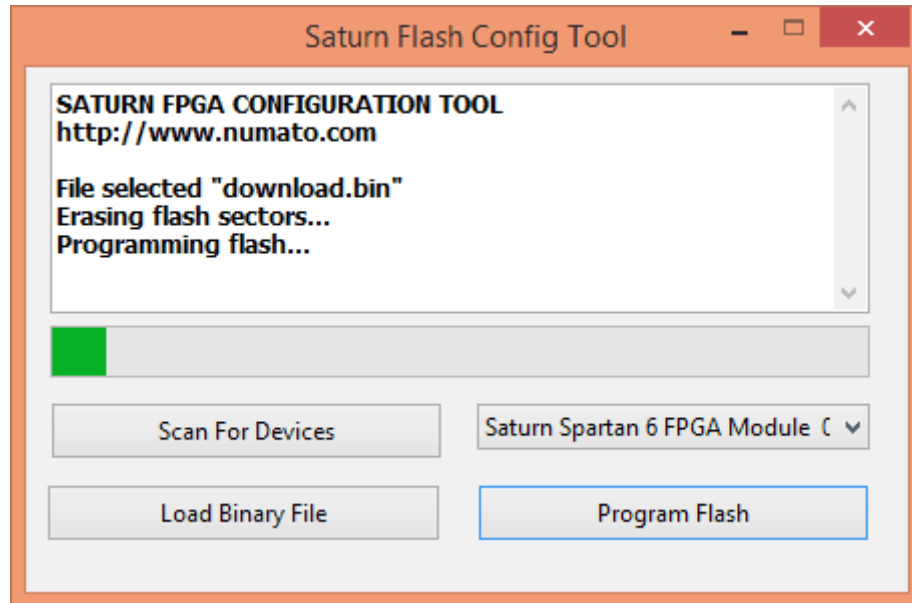
If this command succeeded, you will see a few new files in the directory and one of them would be "download.bin". This is the final binary image that we need to download to Saturn's SPI flash. The following image shows all files in my directory after executing all above mentioned steps.



(https://numato.com/help/wp-content/uploads/2016/03/allimages_afterbuild1.png)

The final image is approximately 10MB in size and this leaves with approximately 6MB of available space in the SPI flash for custom user data.

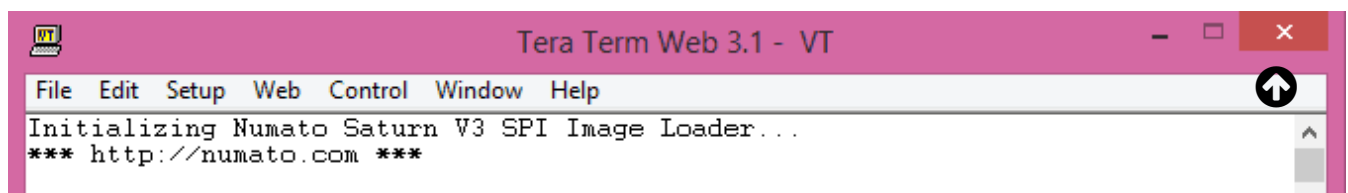
Now download the latest Saturn Spartan 6 FPGA Module configuration tool from the product page (<https://numato.com/saturn-spartan-6-fpga-development-board-with-ddr-sdram>) and use that to download the final binary image "download.bin" to the SPI flash (see image below).



(https://numato.com/help/wp-content/uploads/2016/03/downloading_linux_image1.png)

It may take a couple of minutes for the download to complete. While download is in progress, go ahead and start your favorite serial terminal emulation program (Hyperterminal, PUTTY, TeraTerm etc..) and open the COM port corresponding to Saturn's FT232 channel B and set baud rate to 115200 and set handshaking to off/none (You should have configured channel B of Saturn's FT232H as serial port. If not, please follow this tutorial to do so (<https://numato.com/kb/configuring-ft232h-usb-serial-converter-saturn-spartan-6-module/>)).

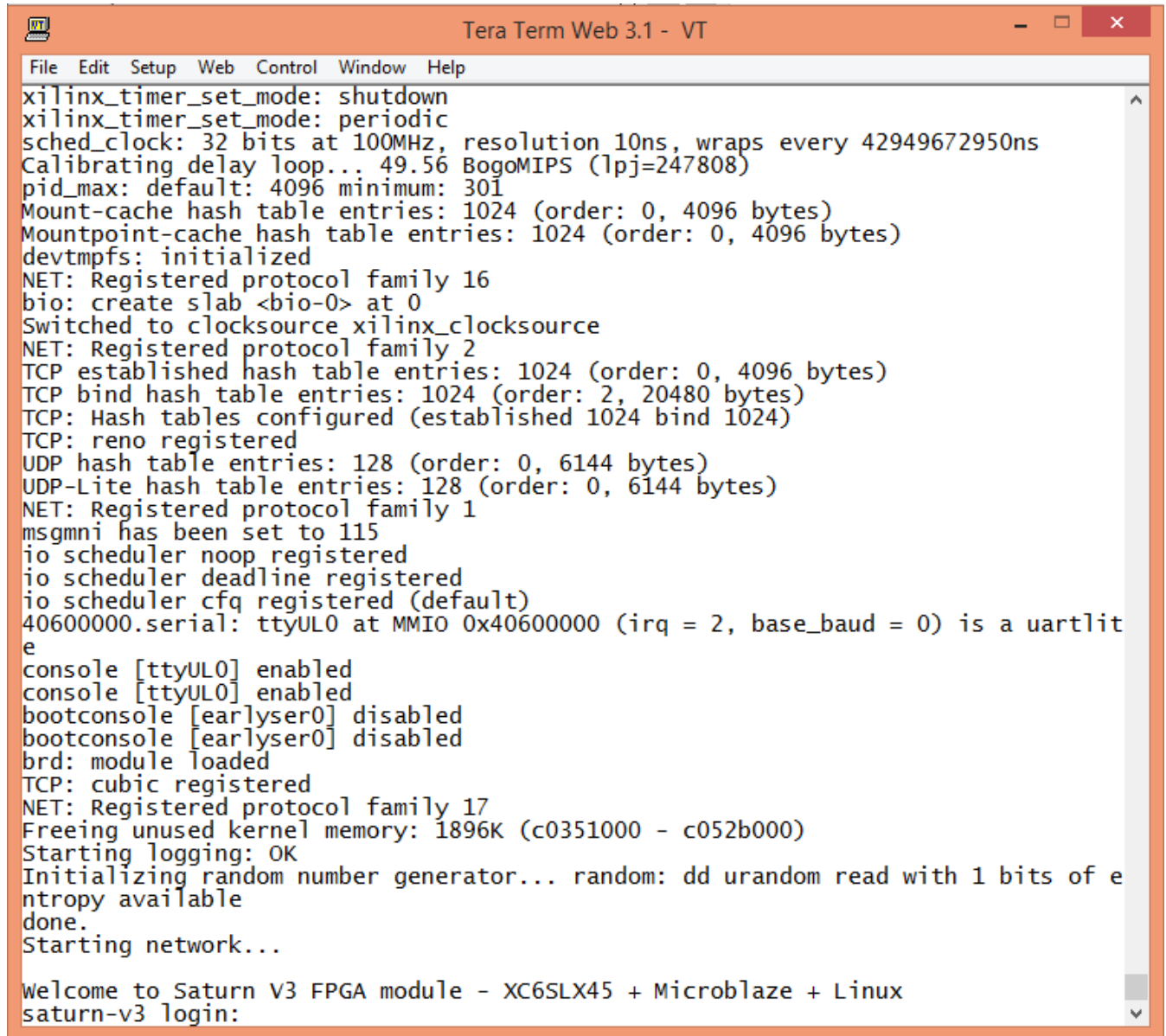
Once the binary image is completed downloading, the tool will reboot the FPGA and let it load the configuration data from newly downloaded image. If everything went well, the bootloader will start copying Linux kernel image to DDR and you will see the below message in the serial terminal emulation software.



Loading application image...

(https://numato.com/help/wp-content/uploads/2016/03/bootloader_loading_linux1.png)

It may take a few seconds for the bootloader to copy the Linux kernel. Once copying is completed, bootloader will start the kernel and you will see the kernel booting as shown below.

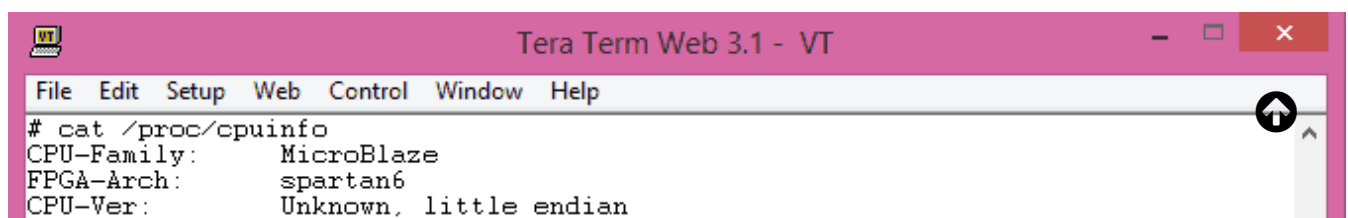
A screenshot of a Tera Term Web 3.1 - VT terminal window. The window has a menu bar with File, Edit, Setup, Web, Control, Window, and Help. The terminal displays a series of Linux boot messages. The messages include: xilinx_timer_set_mode: shutdown, xilinx_timer_set_mode: periodic, sched_clock: 32 bits at 100MHz, resolution 10ns, wraps every 42949672950ns, Calibrating delay loop... 49.56 BogoMIPS (lpj=247808), pid_max: default: 4096 minimum: 301, Mount-cache hash table entries: 1024 (order: 0, 4096 bytes), Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes), devtmpfs: initialized, NET: Registered protocol family 16, bio: create slab <bio-0> at 0, Switched to clocksource xilinx_clocksource, NET: Registered protocol family 2, TCP established hash table entries: 1024 (order: 0, 4096 bytes), TCP bind hash table entries: 1024 (order: 2, 20480 bytes), TCP: Hash tables configured (established 1024 bind 1024), TCP: reno registered, UDP hash table entries: 128 (order: 0, 6144 bytes), UDP-Lite hash table entries: 128 (order: 0, 6144 bytes), NET: Registered protocol family 1, msgmni has been set to 115, io scheduler noop registered, io scheduler deadline registered, io scheduler cfq registered (default), 40600000.serial: ttyUL0 at MMIO 0x40600000 (irq = 2, base_baud = 0) is a uartlite, console [ttyUL0] enabled, console [ttyUL0] enabled, bootconsole [earlyser0] disabled, bootconsole [earlyser0] disabled, brd: module loaded, TCP: cubic registered, NET: Registered protocol family 17, Freeing unused kernel memory: 1896K (c0351000 - c052b000), Starting logging: OK, Initializing random number generator... random: dd urandom read with 1 bits of entropy available, done., Starting network..., Welcome to Saturn V3 FPGA module - XC6SLX45 + Microblaze + Linux, saturn-v3 login:

```
xilinx_timer_set_mode: shutdown
xilinx_timer_set_mode: periodic
sched_clock: 32 bits at 100MHz, resolution 10ns, wraps every 42949672950ns
Calibrating delay loop... 49.56 BogoMIPS (lpj=247808)
pid_max: default: 4096 minimum: 301
Mount-cache hash table entries: 1024 (order: 0, 4096 bytes)
Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes)
devtmpfs: initialized
NET: Registered protocol family 16
bio: create slab <bio-0> at 0
Switched to clocksource xilinx_clocksource
NET: Registered protocol family 2
TCP established hash table entries: 1024 (order: 0, 4096 bytes)
TCP bind hash table entries: 1024 (order: 2, 20480 bytes)
TCP: Hash tables configured (established 1024 bind 1024)
TCP: reno registered
UDP hash table entries: 128 (order: 0, 6144 bytes)
UDP-Lite hash table entries: 128 (order: 0, 6144 bytes)
NET: Registered protocol family 1
msgmni has been set to 115
io scheduler noop registered
io scheduler deadline registered
io scheduler cfq registered (default)
40600000.serial: ttyUL0 at MMIO 0x40600000 (irq = 2, base_baud = 0) is a uartlite
console [ttyUL0] enabled
console [ttyUL0] enabled
bootconsole [earlyser0] disabled
bootconsole [earlyser0] disabled
brd: module loaded
TCP: cubic registered
NET: Registered protocol family 17
Freeing unused kernel memory: 1896K (c0351000 - c052b000)
Starting logging: OK
Initializing random number generator... random: dd urandom read with 1 bits of entropy available
done.
Starting network...

Welcome to Saturn V3 FPGA module - XC6SLX45 + Microblaze + Linux
saturn-v3 login:
```

(https://numato.com/help/wp-content/uploads/2016/03/SaturnSpartan6_Booting_Linux1.png)

You can login to Linux as root with no password. Image below shows cpuinfo output.

A screenshot of a Tera Term Web 3.1 - VT terminal window. The window has a menu bar with File, Edit, Setup, Web, Control, Window, and Help. The terminal displays the output of the command 'cat /proc/cpuinfo'. The output shows: CPU-Family: MicroBlaze, FPGA-Arch: spartan6, CPU-Ver: Unknown, little endian.

```
# cat /proc/cpuinfo
CPU-Family:      MicroBlaze
FPGA-Arch:       spartan6
CPU-Ver:         Unknown, little endian
```

```

CPU-MHz:      100.00
BogoMips:     49.56
HW:
  Shift:      yes
  MSR:        yes
  PCMP:       yes
  DIV:        yes
  MMU:        3
  MUL:        v2
  FPU:        no
  Exc:        op0x0 unal ill iopb dopb zero
Stream-insns: privileged
Icache:       16kB   line length:   32B
Dcache:       16kB   line length:   16B
Dcache-Policy: write-through
HW-Debug:     yes
PVR-USR1:     00
PVR-USR2:     00000000
Page size:    4096
#

```

(https://numato.com/help/wp-content/uploads/2016/03/saturn_cpufreq1.png)

All files and projects that are used/created while writing this article series is available for download here (<https://github.com/numato/samplecode/tree/master/FPGA/Saturn/SaturnLinux>).



(<https://twitter.com/home?status=https://numato.com/kb/saturn-microblaze-and-linux-how-run-linux-saturn-spartan-6-fpga-module-part-iv/>)

and-linux-how-run-linux-saturn-spartan-6-fpga-module-part-iv/)



(<https://www.facebook.com/sharer/sharer.php?u=https://numato.com/kb/saturn-microblaze-and-linux-how-run-linux-saturn-spartan-6-fpga-module-part-iv/>)



(<https://pinterest.com/pin/create/button/?url=https://numato.com/kb/saturn-microblaze-and-linux-how-run-linux-saturn-spartan-6-fpga-module-part-iv/&media=&description=>)

description=)



(<https://plus.google.com/share?url=https://numato.com/kb/saturn-microblaze-and-linux-how-run-linux-saturn-spartan-6-fpga-module-part-iv/>)

iv/)




(<https://www.linkedin.com/shareArticle?mini=true&url=https://numato.com/kb/saturn-microblaze-and-linux-how-run-linux-saturn-spartan-6-fpga-module-part-iv/>)

<https://numato.com/kb/saturn-microblaze-and-linux-how-run-linux-saturn-spartan-6-fpga-module-part-iv/>)

Was this helpful?



 7 Yes

 No

Leave A Comment

Comment

Name

*

Email

*

Website

POST COMMENT

Knowledge Base Category

Callisto Kintex 7 USB 3.1 FPGA Module (<https://numato.com/kb-category/callisto-kintex-7-usb-3-1-fpga-module/>) (2)

Embedded Linux (<https://numato.com/kb-category/embedded-linux/>) (12)

FPGA Applications (<https://numato.com/kb-category/fpga-applications/>) (13)

Galatea PCI Express Spartan 6 FPGA Module (<https://numato.com/kb-category/galatea-pci-express-spartan-6-fpga-module/>) (4)

Getting Started With FPGA (<https://numato.com/kb-category/getting-started-with-fpga/>)  (32)

Intercore SDK Framework (<https://numato.com/kb-category/intercore-sdk-framework/>) (1)

Mimas A7 Mini FPGA Development Board (<https://numato.com/kb-category/mimas-a7-mini-fpga-development-board/>) (3)

Mimas Artix 7 FPGA Development Board (<https://numato.com/kb-category/mimas-artix-7-fpga-development-board/>) (10)

Narvi Spartan 7 FPGA Module (<https://numato.com/kb-category/narvi-spartan-7-fpga-module/>) (2)

Nereid Kintex 7 PCI Express FPGA Board (<https://numato.com/kb-category/nereid-kintex-7-pci-express-fpga-board/>) (3)

Neso Artix 7 FPGA Module (<https://numato.com/kb-category/neso-artix-7-fpga-module/>) (10)

Opsis: FPGA-based open video platform (<https://numato.com/kb-category/opsis-fpga-based-open-video-platform/>) (4)

Prodigy Series Automation Devices (<https://numato.com/kb-category/prodigy-series-automation-devices/>) (1)

Proteus Kintex 7 FPGA Development Module (<https://numato.com/kb-category/proteus-kintex-7-fpga-development-module/>) (1)

Quick Start Guides (<https://numato.com/kb-category/quick-start-guides/>) (2)

Rhea Device Management Tool (<https://numato.com/kb-category/rhea-device-management-tool/>) (2)

Saturn Spartan 6 FPGA Module (<https://numato.com/kb-category/saturn-spartan-6-fpga-module/>) (7)

Skoll Kintex 7 FPGA Module (<https://numato.com/kb-category/skoll-kintex-7-fpga-module/>) (6)

Styx Xilinx Zynq FPGA Module (<https://numato.com/kb-category/styx-xilinx-zynq-fpga-module/>) (8)

Tagus – Artix 7 PCI Express Development Board (<https://numato.com/kb-category/tagus-artix-7-pci-express-development-board/>) (1)

Telesto MAX 10 FPGA Module (<https://numato.com/kb-category/telesto-max-10-fpga-module/>) (5)

Tenagra FPGA System Management Software (<https://numato.com/kb-category/tenagra-fpga-system-management-software/>) (3)



Theia Android Application (<https://numato.com/kb-category/theia-android-application/>)

(1)

USB GPIO Modules (<https://numato.com/kb-category/usb-gpio-modules/>) (2)

USB Relay Modules (<https://numato.com/kb-category/usb-relay-modules/>) (1)

Vivado Design Suit (<https://numato.com/kb-category/vivado-design-suit/>) (4)

Waxwing Spartan 6 FPGA Development Board (<https://numato.com/kb-category/waxwing-spartan-6-fpga-development-board/>) (3)

White Papers (<https://numato.com/kb-category/white-papers/>) (1)

Working With Xilinx EDK (<https://numato.com/kb-category/working-with-xilinx-edk/>) (1)

XO-Bus Framework (<https://numato.com/kb-category/xo-bus/>) (2)

Don't miss out on new articles! Subscribe to get valuable insights.

Subscribe



(<https://twitter.com/numatolab>)



(<https://www.facebook.com/numato/>)



(<https://www.youtube.com/user/NumatoLab>)



(<https://plus.google.com/+Numatosystems>)

[Privacy Policy](#) [Terms of Use](#)

© 2018 Numato Systems Pvt. Ltd. (<https://numato.com>). All Rights Reserved. [Privacy Policy](#) (<https://numato.com/privacy-policy/>) | [Terms of Use](#) (<https://numato.com/tearm-of-service/>) 