

*****Draft*****

These the first steps to upgrading a RPi3 to a RPi4 project for Ultibo 05/16/20

*****Draft*****

```
diff --git a/th_svd/RPi4/config.txt b/th_svd/RPi4/config.txt
index 2064759..b044adb 100755
--- a/th_svd/RPi4/config.txt
+++ b/th_svd/RPi4/config.txt
@@ -60,6 +60,8 @@ dtparam=audio=on
# Enable DRM VC4 V3D driver on top of the dispmanx display stack
dtoverlay=vc4-fkms-v3d
max_framebuffers=2
+enable_gic=1
+armstub=armstub32-rpi4.bin
```

This is the sha256 sum of the armstub32-rpi4.bin.

a5f806af388b952f768ccb5e01130eb66536f5f2de288335f4c5703de8375b6a armstub32-rpi4.bin

These are firmwire being tested for Rpi4.

d239f344f610ecc005a97110b98f620e9bfe1d6a4923af488e025d2083f95e77 bootcode.bin
7716af32619f0771ee91196f367eea307ba33ce61df5f4675ed3ea0b16d16949 fixup4x.dat
215d6ccdea1e8b62643c626039bfccd751753b1bfe79be269b171ea66134eb2b start4.elf
210478ae179a91d02a3c15232a899215b8bda2e7fe1bb3461b15264f549d391f start4x.elf

Ultibo Wrote

“We just did a test of taking a brand new SanDisk card from the packet, it had a FAT32 partition on it already from the factory, we copied the Ultibo demo image (kernel7l.img) and the Pi 4 firmware files (start4.elf and fixup4.dat) to the card and booted the Pi from it”

With great power comes great responsibility, so there are a few things you need to remember.

- All Pi 4 models include two different interrupt controllers, which one is enabled by default is slightly dependant on the firmware version, the presence of device tree and the config.txt settings.

In order to ensure that Ultibo kernels always boot we have gone to great lengths to detect which interrupt controller is active and adjust accordingly, however the two are not completely equal so when reverting to the so called legacy controller some features are unavailable.

To be sure you are always using the new GIC controller your config.txt file must contain the line **enable_gic=1**

- Some changes to the Pi firmware late last year made it impossible to switch back to secure mode during boot, while that doesn't cause much problem on earlier models on the Pi 4 the inability to access secure mode means fast interrupts (FIQ) are unavailable.

We've provided a solution in the form of a custom boot stub that the Ultibo boot code looks for in order to determine if it can return to secure mode, to make this work you need to ensure that your SD card includes the relevant **armstub32-rpi?.bin** file plus the appropriate **armstub=** value in the config.txt file.

To make this as easy as possible we've included the new boot stubs and an example config.txt in the firmware folder of the Ultibo installation (C:\Ultibo\Core\firmware on Windows) along with copies of the latest available firmware for all models.

- The Pi 4 increased the rate of the core oscillator from 19.2MHz to 54MHz, in the past we've always set the dividers so that the system clock ticks at a rate of 1MHz which has the side benefit that it represents 1 microsecond per tick.

For the Pi 4 we've chosen to allow the clock to run at the full 54MHz rate, this means if you have used the ClockGetTotal function to provide a counter in your application and rely on the count being in microseconds that will no longer be true for the Pi 4.

The solution is very simple, either use GetTickCount64 from the RTL instead which will provide a millisecond counter or divide the value returned from ClockGetTotal by the value of the system variable CLOCK_CYCLES_PER_MICROSECOND to obtain microseconds.

We're sure you'll have a bunch of questions once you start to explore the new release, as always feel free to ask.

This project as Rpi3 used uTFTP.pas to transfer the kernel7.img to Ultibo system.

```
a5f806af388b952f768ccb5e01130eb66536f5f2de288335f4c5703de8375b6a armstub32-rpi4.bin
f1f5f6a47af524c947c40bbbf9d6d292663e1d5c27bd2f38b2ebce629cf6d476 blu.pgm
d239f344f610ecc005a97110b98f620e9bfe1d6a4923af488e025d2083f95e77 bootcode.bin
4fe93f768a53a54ac1d54561ffaa954f24c9140b3b1eca35e2fd2b620b81f746 config.txt
7716af32619f0771ee91196f367eea307ba33ce61df5f4675ed3ea0b16d16949 fixup4x.dat
acfce694ade49c8ec20267f30f5015b0ef747ab16f1517c15c00af92fa242f51 fixup_x.dat
c1cb4703861b7b2bcf9d45947143a6e06da8c40e7f01cf1dc2ed122752658abc fixup_x.dat
c2a63ec4076ffe9ed81f5d0ab6f9bda85970506c7a975e4d8af2845139a4bf8c grn.pgm
3f9a20cbd723fcf1006da0800f376bd308f76569e20efd2b8885592200ba3b64 kernel7l.img
6a180d44ce2d6b8d1e95781a10b3490f6205cdc3c4a85d816346237b34f669b9 red.pgm
215d6ccdea1e8b62643c626039bfccd751753b1bfe79be269b171ea66134eb2b start4.elf
210478ae179a91d02a3c15232a899215b8bda2e7fe1bb3461b15264f549d391f start4x.elf
8ca471067f496452b24cb492919b5d789681bf77e5ea97c04033f7d29c7e8ac7 start.elf
ec57fb988543b951413e8f4262cab2aeb1475c533716cf8878f88375fdcccbe7 start_x.elf
```

The installer script is found at "<https://raw.githubusercontent.com/ultibohub/Tools/master/Installer/Core/Linux/ultiboinstaller.sh>".

Note: Be sure and copy the raw from GitHub into a file ultiboinstaller.sh.

```
-rwxr-xr-x 1 devel devel 26813 May 15 06:48 ultiboinstaller.sh
md5sum ultiboinstaller.sh
85dc58dff3bdcd0de510d69b0456ea60 ultiboinstaller.sh
```

./

Linux installer for Free Pascal and Lazarus (Ultibo edition)

This installation will download the sources for:

- Ultibo core
- Ultibo examples
- Free Pascal (Ultibo edition)
- Lazarus IDE (Ultibo edition)

Then it will build all of the above, this will take several minutes to complete depending on the speed of your system.

The installation will not interfere with any existing development environments including other installations of Free Pascal and Lazarus.

Continue (y/n)?

Do you want to build and install the Lazarus IDE (y/n)?

Free Pascal and Lazarus (Ultibo edition) prerequisites

Installing and building Free Pascal requires several tools from the build essentials package including make, ld and as as well as the unzip utility.

These can be installed on Debian based distributions using:

```
sudo apt-get install build-essential unzip
```

Lazarus IDE requires the GTK2 and X11 dev packages which can be installed on Debian based distributions by using:

```
sudo apt-get install libgtk2.0-dev libcairo2-dev \
  libpango1.0-dev libgdk-pixbuf2.0-dev libatk1.0-dev \
  libghc-x11-dev
```

Cross compiling Ultibo applications from Linux requires the arm-none-eabi build of the binutils package, this can be installed on Debian based distributions using:

```
sudo apt-get install binutils-arm-none-eabi
```

Press return to check for these prerequisites

Enter an installation folder or press return to accept the default install location

[/home/devel/ultibo/core]:

The install folder will be:
/home/devel/ultibo/core

Continue? (y,n):

After install do you want a shortcut created in:
/home/devel/.local/share/applications (y/n)?

Do you want to build the Hello World examples (y/n)?

This project uses several C files in addition to Pascal files

disp_mat.c master.c mythread.c svd.c
error.c mul_mat.c pnmio.c trans_mat.c

There is buildlib.sh that compiles into libsvd.a

devel@mypi3-20:~/Ultibo_Projects/th_svd/RPi4 \$./buildlib.sh
Compiling example ultibo_th_svd

```
#!/bin/bash
#export PATH=/home/devel/ultibo/core/fpc/bin:$PATH
rm -f *.o
rm -f libsvd.a
```

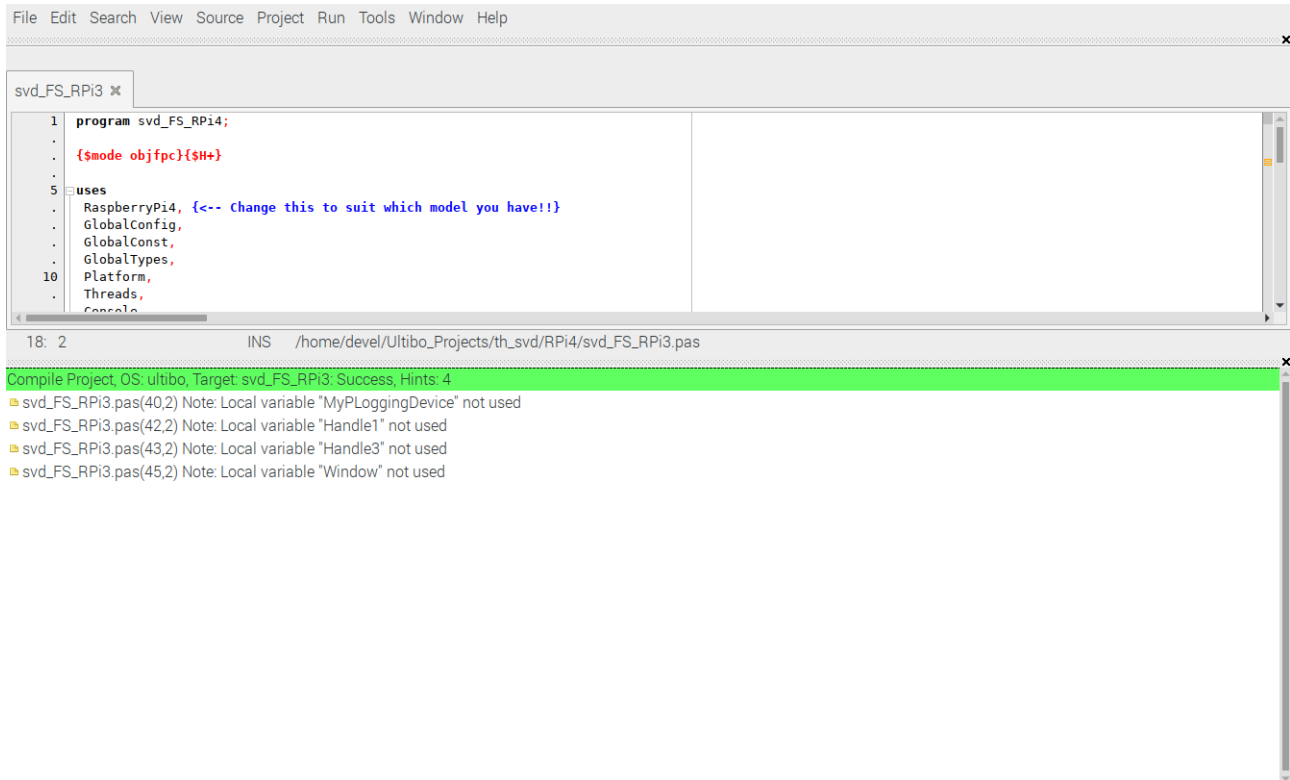
```
arm-none-eabi-gcc -I../include -O3 -mabi=aapcs -marm -march=armv7-a -mfpu=vfpv3-d16 -
mfloat-abi=hard -c svd.c -o svd.o
arm-none-eabi-gcc -I../include -O3 -mabi=aapcs -marm -march=armv7-a -mfpu=vfpv3-d16 -
mfloat-abi=hard -c disp_mat.c -o disp_mat.o
arm-none-eabi-gcc -I../include -O3 -mabi=aapcs -marm -march=armv7-a -mfpu=vfpv3-d16 -
mfloat-abi=hard -c trans_mat.c -o trans_mat.o
arm-none-eabi-gcc -I../include -O3 -mabi=aapcs -marm -march=armv7-a -mfpu=vfpv3-d16 -
mfloat-abi=hard -c mul_mat.c -o mul_mat.o
arm-none-eabi-gcc -D_POSIX_THREADS -lpthread -I../include -O3 -mabi=aapcs -marm -
march=armv7-a -mfpu=vfpv3-d16 -mfloat-abi=hard -c mythread.c -o mythread.o
arm-none-eabi-gcc -I../include -O3 -mabi=aapcs -marm -march=armv7-a -mfpu=vfpv3-d16 -
mfloat-abi=hard -c pnmio.c -o pnmio.o
arm-none-eabi-gcc -I../include -O3 -mabi=aapcs -marm -march=armv7-a -mfpu=vfpv3-d16 -
mfloat-abi=hard -c error.c -o error.o
```

```
echo "Compiling example ultibo_th_svd "
arm-none-eabi-gcc -DUltibo -D_POSIX_THREADS -lpthread -I../include -O3 -mabi=aapcs -marm
-march=armv7-a -mfpu=vfpv3-d16 -mfloat-abi=hard -c master.c -o ultibo_th_svd.o
```

```
#gcc test_svd.c svd.o disp_mat.o -lm -o test_svd
arm-none-eabi-ar rcs libsvd.a *.o
arm-none-eabi-ar -t libsvd.a > libsvd_obj.txt
#fpc -vi -B -Tultibo -Parm -CpARMV7A -WpRPi3B @/home/devel/ultibo/core/fpc/bin/RPi3.CFG
-O4 svd_FS_Rpi3.lpr
```

The objects in libsvd.a are found in listed in the file libsvd_obj.txt

disp_mat.o
error.o
mul_mat.o
mythread.o
pnmio.o
svd.o
trans_mat.o
ultibo_th_svd.o



The screenshot shows a code editor window with a menu bar (File, Edit, Search, View, Source, Project, Run, Tools, Window, Help) and a tab labeled 'svd_FS_RPi3'. The code is in Pascal and defines a program 'svd_FS_RPi4'. It includes a comment to change the RaspberryPi4 model. The 'uses' section lists several units: RaspberryPi4, GlobalConfig, GlobalConst, GlobalTypes, Platform, Threads, and Console. The status bar at the bottom indicates the cursor is at line 18, column 2 in the file '/home/devel/Ultibo_Projects/th_svd/RPi4/svd_FS_RPi3.pas'. Below the editor, a green bar shows the compilation status: 'Compile Project, OS: ultibo, Target: svd_FS_RPi3: Success, Hints: 4'. Four yellow warning icons are listed, each noting that a local variable ('MyLoggingDevice', 'Handle1', 'Handle3', and 'Window') was not used.

```
1  program svd_FS_RPi4;  
  .  
  .  
  .  
5  uses  
  .   RaspberryPi4, {<-- Change this to suit which model you have!!}  
  .   GlobalConfig,  
  .   GlobalConst,  
  .   GlobalTypes,  
10  .   Platform,  
  .   Threads,  
  .   Console;  
18: 2
```

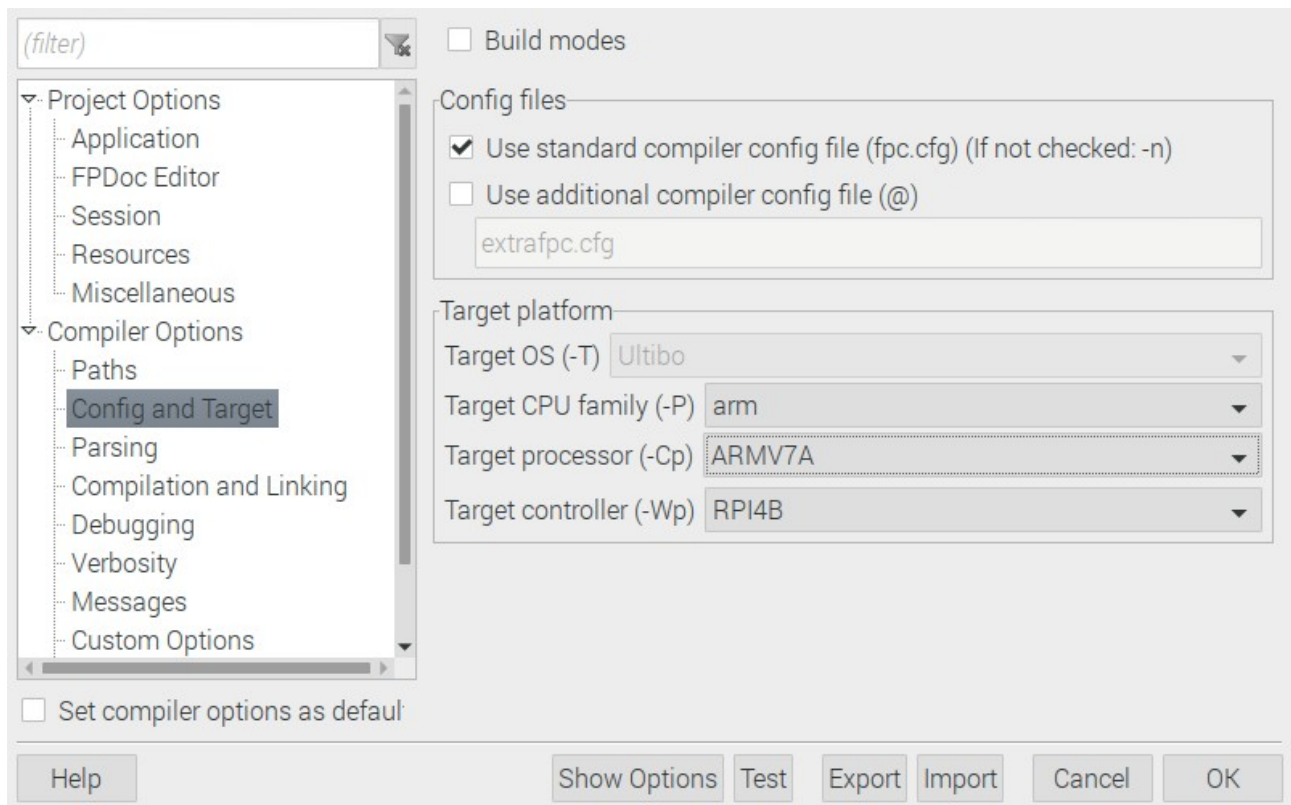
INS /home/devel/Ultibo_Projects/th_svd/RPi4/svd_FS_RPi3.pas

Compile Project, OS: ultibo, Target: svd_FS_RPi3: Success, Hints: 4

- svd_FS_RPi3.pas(40,2) Note: Local variable "MyLoggingDevice" not used
- svd_FS_RPi3.pas(42,2) Note: Local variable "Handle1" not used
- svd_FS_RPi3.pas(43,2) Note: Local variable "Handle3" not used
- svd_FS_RPi3.pas(45,2) Note: Local variable "Window" not used

Depressing the Run/Compile creates the kernel7l.img.
-rwxr-xr-x 1 devel devel 3067936 May 15 09:24 kernel7l.img

project options



Depressing the Run/Compile creates the kernel7l.img