Building Image in Raspberry pi 3 with using yocto-project.

** **Raspberry pi 3 specifications** **
SOC: Broadcom BCM2837
CPU: 4* ARM Cortex-A53, 1.2GHz (1.2GHz is a clock speed of processor)
GPU: Broadcom Video Core IV
RAM: 1GB LPDDR2 (900MHZ)
Networking : 10/100 Ethernet , 2.4 GHz 802.11n wireless.
Bluetooth: Bluetooth 4.1 Classic, Bluetooth Low energy
Storage Type: use external SD-card or have emmc storage.
GPIO Pins: 40 pins available
Ports: HDMI, RJ45, USB, CSI (camera serial interface), DSI (display serial interface).

** **Hardware support of any raspberry pi board** **
Need to open embedded layer.
Need to open embedded core layer.
Need to open embedded multimedia layer.
Need to open embedded networking layer.
Need to open embedded python layer.
Note: "Raspberry pi supported own BSP layer to build a perfect image which is easily run on raspberry pi boards"

** **Supported Hardware boards** **
Raspberry pi
Raspberry pi 2
Raspberry pi 3
Raspberry pi 4
Raspberry Zero
Raspberry Zero wireless

** **Follow the procedure to create an image on ras pi 3board with yocto-project** **

step 1 : Create a new folder in your linux terminal and write name is "raspi3".

```
$ mkdir raspi3
```

step 2 : Go to this folder

```
$ cd raspi3
```

step 3 : Time to clone the bsp layer of raspi 3 .
open chrome browser
enter this link in your browser >> https://layers.openembedded.org/layerindex/branch/master/layers/
click on the machine section
search for raspberrypi3
select the 64 bit raspi3 bsp layer and click on meta-raspberrypi
Copy the git link which is available in this page and clone this link in your created directory.

```
$ git clone git://git.yoctoproject.org/meta-raspberrypi
```

step 4 : After clone is successfully you are also add in this folder poky layer also.

```
$ git clone git://git.yoctoproject.org/poky
```

Step 5 : After poky clone is successful then you are also clone openembedded respository in same folder.

```
$ git clone git://git.openembedded.org/meta-openembedded.
```

Step 6 : all clone is successfully completed in your raspi3 folder , so it look like this types

```
$ ls –l
```

folder 1 >> **poky**
folder 2 >> **meta-raspberrypi**
folder 3 >> **meta-openembedded**

Step 7 Now set the open-embedded build environment in poky folder.
Enter in poky folder.
run this command.

```
$ source oe-init-build-env ../build/
```

notes: it means it creates build folder outside of the poky folder.

Step 7 Now check the how many layers are present in your bblayers.conf file and this command write in build folder.

```
$ bitbake-layers show-layers
```

** it show only three layers like meta, meta-poky, meta-yocto-bsp.

Step 8 Add the meta-raspberry pi layers in bblayers.conf file so write this command in the build folder.

```
$ bitbake-layers add-layer ../meta-raspberrypi/
```

** If you check again layers list then you show meta-raspberrypi layers.

```
$ bitbake-layers show-layers
```

Step 9 Add the meta-oe layers in bblayers.conf file so write this command in the build folder.

```
$ bitbake-layers add-layer ../meta-openembedded/meta-oe
```

** If you check again layers list then you show meta-oe layers.

```
$ bitbake-layers show-layers
```

Step 10 Add the meta-python layers in bblayers.conf file so write this command in the build folder.

```
$ bitbake-layers add-layer ../meta-openembedded/meta-python
```

** If you check again layers list then you show meta-python layers.

```
$ bitbake-layers show-layers
```

Step 11 Add the meta-multimedia layers in bblayers.conf file so write this command in the build folder.

```
$ bitbake-layers add-layer ../meta-openembedded/meta-multimedia
```

** If you check again layers list then you show meta-multimedia layers.

```
$ bitbake-layers show-layers
```

Step 12 Add the meta-networking layers in bblayers.conf file so write this command in the build folder.

```
$ bitbake-layers add-layer ../meta-openembedded/meta-networking
```

** If you check again layers list then you show meta-networking layers.

```
$ bitbake-layers show-layers
```

Step 13 Check the how many hardware machine supported in meta-raspberry pi layers

```
$ cd meta-raspberrypi/conf/machine
$ ls
```

** show all machine list but iam used and build raspberrypi3 module. You are use only name of the machine.
for example **raspberrypi3-64.conf** **(**raspberrypi3-64 is a machine name not used .conf file).

Step 14 go to the build folder and enter conf folder and then open local.conf file.

$ vim local.conf

Add this line in the end line of this file to configure rasphberry pi3 modules.

MACHINE = "raspberrypi3"

INHERIT += "rm_work"

ENABLE_UART = "1"

Step 15 Now check the which types of images are present in the raspberrypi3 module

cd meta-raspberrypi

cd recipes-core

cd images

** check the image of your images.

for example :: **rpi-test-image**.bb

rpi-test-image this is the image name which you build by using bitbake command.

Step 16 Now switch to all the folder in same branch first check it and it work fine in "zeus" branch or release.

$ cd poky

$ git branch >> if you show master branch name then you will select first zeus branch.

$ git checkout zeus

Step 17 Now switch to all the folder in same branch first check it and it work fine in "zeus" branch or release.

$ cd meta-raspberrypi

$ git branch >> if you show master branch name then you will select first zeus branch.

$ git checkout zeus

Step 18 Now switch to all the folder in same branch first check it and it work fine in "zeus" branch or release.

$ cd meta-openembedded

$ git branch >> if you show master branch name then you will select first zeus branch.

$ git checkout zeus

Step 19 Again go to the Poky folder and set the environment.

$ cd poky

$ source oe-init-build-env ../build/

$ bitbake rpi-hwup-image

or

$ bitbake rpi-test-image

Step 20 Now finally starting the build process and parses the recipes through bitbake in your adding layers and match the configuration and classes files.

Step 21 After the build is successfully go to the image folder.

$ cd build

$ cd tmp/deploy/images/raspberrypi3/

* your image name is "**rpi-hwup-image-raspberrypi3.rpi-sdimg**"

or

* your image name is "**rpi-test-image-raspberrypi3.rpi-sdimg**"

Step 22 Insert the card reader and min 16 gb memory card in your host machine.

Step 23 check the status of your sd-card insert or not.

```
$ lsblk        ( show your sd-card devices name)
```

Step 24 First format your sd-card and remove all the partition if created in your sdcard .

Step 25 Now again check your sd-card has proper format or present no more partition.

```
$ lsblk
```

Step 26 enter this command in your raspberry pi image folder .

```
$ sudo dd if=rpi-hwup-image-raspberrypi3.rpi-sdimg of=/dev/sdb bs=4096 status=progress && sync
```

dd = disk drive
if = input file
of = output file
sync = finish the process and ready image into the card

Step 27 remove the card –reader

Step 28 Again insert the card-reader in the host machine and verify the partition of your images present in the sd-card.

```
$ sudo fdisk  /dev/sdb
```

enter p to check the patition table
enter q for quit.

** Booting process in Raspberry Pi 3 **

S1 = GPU core
S2 = First stage bootloader(FSBL) , which is stored in ROM on the SOC(System on chip).
S3 = bootcode.bin
S4 = start.elf
S5 = config.txt
S6 = cmdline.txt
S7 = kernel.img
S8 = image is start

** Hardware Setup **

* An Sd-card with image flashed
* Raspberry pi 3
* A power adapter that can supply 5V
* USB TTL-2303(PL2303) for serial communication.
* USB TTL is connected to the j8 connector of Raspberry Pi3 in the following formation.

* J8 pin	* USB TTL Function	
6	GND Ground	
8	RXL	
10	TXL	(Also check output in minicom CONSOLE)