Working with NVIDIA Tegra BSP and Supporting Latest CUDA Versions

Leon Anavi
Konsulko Group
leon.anavi@konsulko.com
leon@anavi.org
Yocto Project Summit 2019

Konsulko Group

- Services company specializing in Embedded Linux and Open Source Software
- Hardware/software build, design, development, and training services
- Based in San Jose, CA with an engineering presence worldwide
- http://konsulko.com/

Agenda

Konsulko Group

- Introduction to CUDA, Jetson and Tegra
- Challenges in supporting CUDA for the Yocto Project and OpenEmbedded
- Solutions for building custom Linux distributions with latest CUDA version with Yocto Projects and OpenEmbedded
- Flashing, booting and testing
- Conclusions

CUDA, Jetson & Tegra

Konsulko Group

CUDA

Konsulko Group

- **C**ompute **U**nified **D**evice **A**rchitecture
- Parallel computing platform and programming model developed by NVIDIA that allows seamlessly to use GPU for general purpose computing
- Appropriate for 3D graphics as well as a wide range of machine learning and artificial intelligence (AI) applications
- Initially released in 2007

NVIDIA Jetson

Konsulko Group

- Leading AI computing platform for GPU-accelerated parallel processing with CUDA on mobile and embedded devices
- Bringing the power of modern AI to embedded systems with ARM CPUs for robotics and autonomous machines.
- Jetson models feature the Tegra ARM SoC

NVIDIA Jetson

Konsulko Group

JETSON	CUDA	AVAILABLE THROUGH
TK1	6.5	Jan 2024 (for the CPU) / 2025 for Toradex Apalis TK1
TX1	10	Jan 2021
TX2	10	Apr 2022
TX2i	10	April 2028
Xavier	10	Jan 2025
Nano	10	Jan 2025

NVIDIA L4T (Linux for Tegra)

Konsulko Group

- Part of NVIDIA JetPack SDK
- Uses NVIDIA SDK Manager with graphical user interface
- Includes a reference filesystem derived from Ubuntu 18.04
- For more details:
  https://developer.nvidia.com/embedded/linux-tegra
  https://developer.nvidia.com/embedded/jetpack

CUDA & GCC/G++

Konsulko Group

CUDA VERSION	REQUIRED GCC/G++ VERSION
10	7
9	6
8	5
7	5
6.5	4.8

Building a custom GNU/Linux distribution with Yocto Project & OpenEmbedded for Tegra

Konsulko Group

Benefits

Konsulko Group

Benefits for using the Yocto Project and OpenEmbedded instead of L4T reference filesystem derived from Ubuntu:

- Custom distribution based on Poky, the reference distribution of the Yocto Project, for the exact needs of particular embedded device
- Flexibility to select init system, display server protocol, desktop environment, etc.
- Option to integrate software over the air update mechanism

BSP Layers

Konsulko Group

- meta-tegra
  https://github.com/madisongh/meta-tegra
- meta-jetson-tk1
  https://github.com/cubicool/meta-jetson-tk1
- meta-toradex-tegra
  http://git.toradex.com/cgit/meta-toradex-tegra.git

Yocto Project Releases

Konsulko Group

YOCTO PROJECT RELEASE CODENAME	YOCTO PROJECT RELEASE VERSION	YOCTO PROJECT RELEASE DATE	MINIMAL SUPPORTED GCC VERSION
Warrior	2.7	April 2019	8.3
Thud	2.6	November 2018	7.3
Sumo	2.5	April 2018	7.3
Rocko	2.4	October 2017	6.4
Pyro	2.3	May 2017	5.4
Morty	2.2	November 2016	5.4
Krogoth	2.1	April 2016	4.9
Jethro	2	November 2015	4.8

Meta-tegra

Konsulko Group

- Yocto/OE BSP for Tegra devices
- Hosted in GitHub under MIT License
- Started by Matt Madison
- More than 700 commits by 16 contributors
- https://github.com/madisongh/meta-tegra

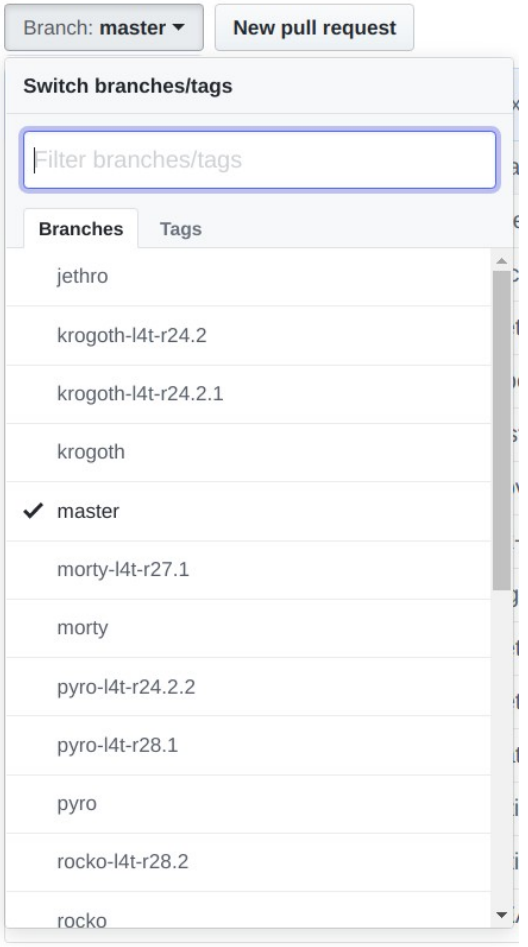
Meta-tegra Supported Devices

Konsulko Group

- Jetson-TX1 development kit (Linux4Tegra R32.2.1, JetPack 4.2.2)
- Jetson-TX2 development kit (Linux4Tegra R32.2.1, JetPack 4.2.2)
- Jetson AGX Xavier development kit (Linux4Tegra R32.2, JetPack 4.2.2)
- Jetson Nano development kit (Linux4Tegra R32.2.1, JetPack 4.2.2)
- Jetson-TX2i module (Linux4Tegra R32.2.1, JetPack 4.2.2)
- Jetson-TX2 4GB module (Linux4Tegra R32.2.1, JetPack 4.2.2)

Meta-tegra Releases

- Jethro
- Krogoth
- Morty
- Rocko
- Sumo
- Thud
- Warrior
- Master (to become Zeus)



Getting Started (1/2)

Konsulko Group

- Recommended host distribution is Ubuntu
- Since JetPack 4.2 manual download through NVIDIA SDK Manager with NVIDIA Developer Network login is required
- Set the path to the downloaded packages in variable **NVIDIA_DEVNET_MIRROR**, for example:

```
NVIDIA_DEVNET_MIRROR = "file:///home/leon/Downloads/nvidia/sdkm_downloads"
```

- SDK Manager downloads a different CUDA package depending on the Ubuntu version, by default for 18.04. If you are using 16.04 set:

```
CUDA_BINARIES_NATIVE = "cuda-binaries-ubuntu1604-native"
```

Getting Started (2/2)

Konsulko Group

- Set machine, for example:

```
MACHINE = "jetson-tx2"
```

- Whitelist commercial license flags:

```
LICENSE_FLAGS_WHITELIST = "commercial"
```

- Generate script for flashing bootloader, rootfs, and other necessary artifacts to the on-board eMMC of Jetson Developer kit:

```
IMAGE_CLASSES += "image_types_tegra"
IMAGE_FSTYPES = "tegraflash"
```

Putting the Pieces Together

Konsulko Group

CUDA VERSION	REQUIRED GCC/G++ VERSION	LATEST YOCTO/OE RELEASE WITH THIS GCC/G++ VERSION
10	7	Thud (2.6)
9	6	Rocko (2.4)
8	5	Pyro (2.3)
7	5	Pyro (2.3)
6.5	4.8	Jethro (2)

The big problem: How to use the latest CUDA on the latest release of the Yocto Project?

Konsulko Group

Solutions

Konsulko Group

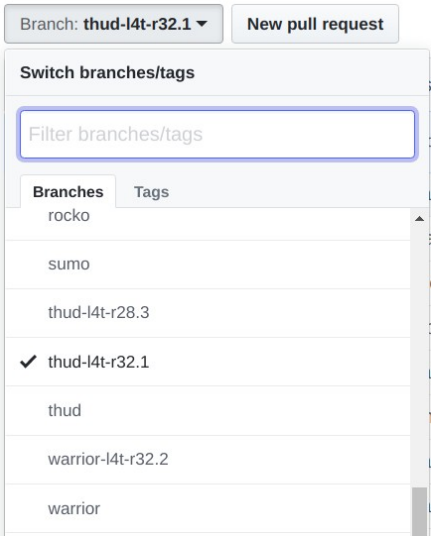
- Use the newest release of the Yocto Project with appropriate GCC version, for example CUDA 10 on Thud with GCC 7
- Use an external toolchain with the appropriate GCC version
- Use backported GCC version on the latest release of the Yocto Project, for example GCC 7 from **meta-tegra/contrib**

Solution 1: Older Compatible Release

Konsulko Group

- Identify a release of the Yocto Project with contains recipes for a compatible GCC version supported by the targeted CUDA version, for examples CUDA 10 requires GCC 7 which is available in release Thud (2.6)

CUDA VERSION	REQUIRED GCC/G++ VERSION	LATEST YOCTO/OE RELEASE WITH THIS GCC/G++ VERSION
10	**7**	**Thud (2.6)**
9	6	Rocko (2.4)
8	5	Pyro (2.3)
7	5	Pyro (2.3)
6.5	4.8	Jethro (2)



- https://www.konsulko.com/building-a-custom-linux-distribution-for-nvidia-cuda-enabled-embedded-devices/

Solution 2: External Toolchain (1/2)

Konsulko Group

- Download a compatible external toolchain, for example from Linaro or ARM
- Use Yocto/OE layer **meta-linaro/meta-linaro-toolchain** https://git.linaro.org/openembedded/meta-linaro.git/about/
- Set external toolchain in **conf/local.conf**:

```
TCMODE = "external-linaro"
EXTERNAL_TOOLCHAIN = "/opt/toolchains/gcc-x86_64_aarch64-linux-gnu"
```

- Disable unsupported by GCC 7 flags

```
DEBUG_PREFIX_MAP = "-fdebug-prefix-map=${WORKDIR}=/usr/src/debug/${PN}/$
{EXTENDPE}${PV}-${PR} \
                -fdebug-prefix-map=${STAGING_DIR_HOST}= \
                -fdebug-prefix-map=${STAGING_DIR_NATIVE}= \

"
```

Solution 2: External Toolchain (2/2)

Konsulko Group

- Replace **--enable-default-pie** with –**disable-pie,** set by security_flags.inc in Poky, to successfully configure and compile **native-glibc** if you plan to build an SDK

```
GCCPIE ?= "--disable-pie"
```

- Create **.bbappend** files that remove unsupported flags from CPPFLAGS, for example **missing-attributes**, for all recipes failing to build with an older compiler due to similar issues

```
cc1: error: -Werror=missing-attributes: no option -Wmissing-attributes
```

- For more details:

https://github.com/madisongh/meta-tegra/wiki/Using-linaro-gcc7-for-CUDA-support

Solution 3: Backported GCC 7

Konsulko Group

- Yocto/OE layer meta-tegra branch master contains a sublayer **contrib** with backported recipes for GCC/G++ 7
- Add layers to **conf/bblayers.conf**
- Specify GCC and CUDA versions in **conf/local.conf**:

```
GCCVERSION = "7.%"
require contrib/conf/include/gcc-compat.conf

CUDA_VERSION="10.0"
CUDA_BINARIES_NATIVE = "cuda-binaries-ubuntu1604-native"
```

One more thing...

Konsulko Group

- Add sample applications to be able to quickly verify that CUDA is properly running on the devices after booting
- For example, add to **conf/local.conf**:

```
IMAGE_INSTALL_append = " cuda-samples"
```

Testing

Konsulko Group

Flashing an image

Konsulko Group

- Enter recovery mode on Jetson developer kit by holding the recovery button (REC) and press reset (RST) **or** by interrupting u-boot and executing:

```
enterrcm
```

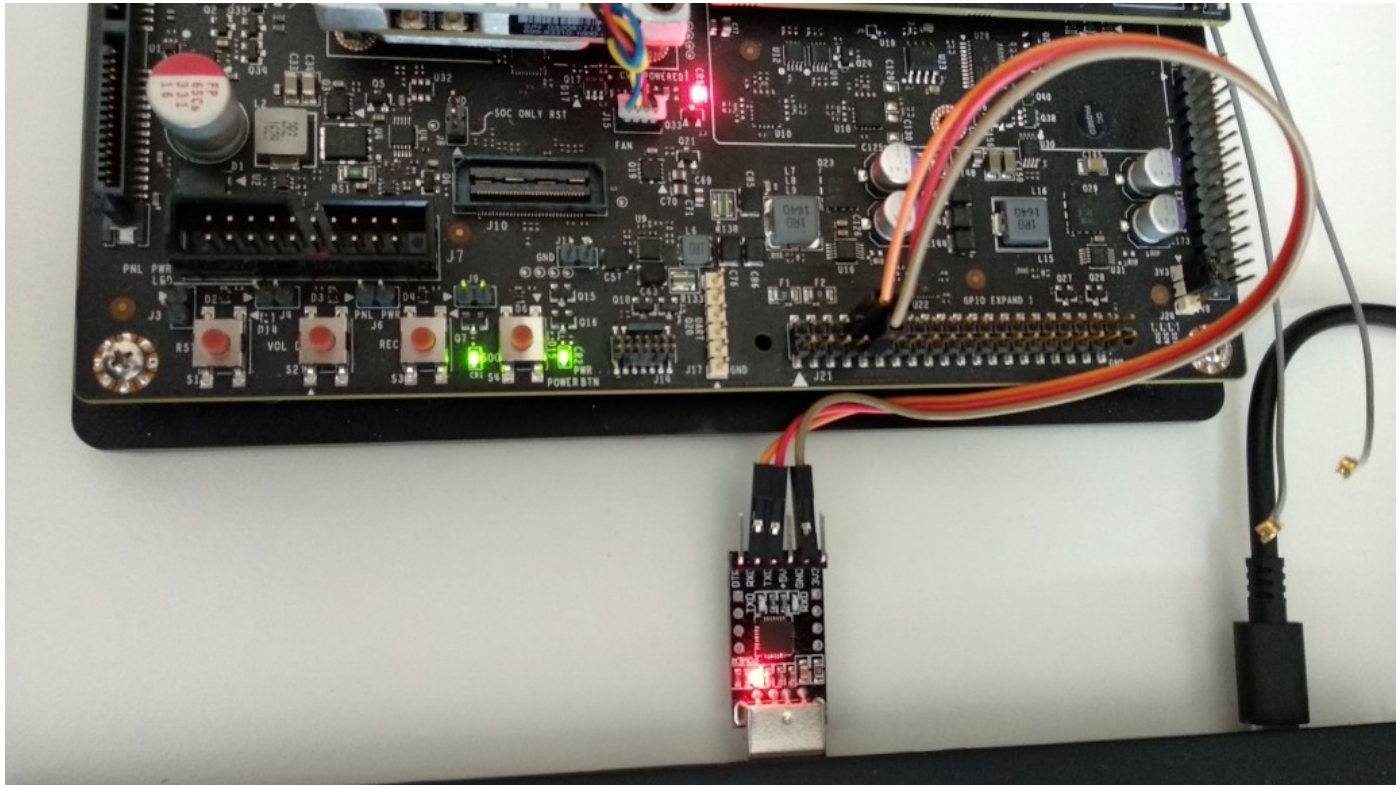
- Connect Jetson dev kit to the build machine with USB cable
- Extract the archive created by bitbake and flash it:

```
tegradir=$(mktemp -u /tmp/tx2.XXXXXXXX)
unzip tmp/deploy/images/jetson-tx2/core-image-minimal-jetson-tx2.tegraflash.zip -d $tegradir
cd $tegradir
sudo ./doflash.sh
```

Booting on Jetson TX2

Konsulko Group

- USB to UART cable can be attached to pins 8 (TX), 9 (GND) and 10 (RX) of J21 on Jetson TX2 developer kit



Testing CUDA

Konsulko Group

- Run **/usr/bin/cuda-samples/deviceQuery**
- If everything works as expected it will print **Result = PASS**
- After that run the other sample application **/usr/bin/cuda-samples/UnifiedMemoryStreams**

```
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            Yes
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 0 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simu
ltaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 10.0, CUDA Runtime Vers
ion = 10.0, NumDevs = 1
Result = PASS
root@jetson-tx2:~#
```

```
Task [18], thread [3] executing on device (741)
Task [19], thread [1] executing on device (841)
Task [20], thread [0] executing on device (942)
Task [21], thread [2] executing on device (711)
Task [22], thread [3] executing on device (718)
Task [23], thread [1] executing on device (504)
Task [24], thread [0] executing on device (822)
Task [25], thread [2] executing on device (833)
Task [26], thread [3] executing on device (406)
Task [27], thread [1] executing on device (600)
Task [28], thread [0] executing on host (64)
Task [29], thread [2] executing on device (750)
Task [30], thread [3] executing on device (844)
Task [31], thread [1] executing on device (267)
Task [32], thread [0] executing on device (648)
Task [33], thread [2] executing on device (262)
Task [34], thread [3] executing on device (968)
Task [35], thread [1] executing on device (256)
Task [36], thread [0] executing on host (64)
Task [37], thread [2] executing on device (340)
Task [38], thread [3] executing on device (711)
Task [39], thread [1] executing on device (947)
All Done!
root@jetson-tx2:~#
```

Conclusions

Konsulko Group

Conclusions

Konsulko Group

- Yocto/OE layer meta-tegra provides the required BSP for building custom embedded distributions for modern Tegra devices
- Latest CUDA version requires older GCC/G++ version which are not available in the latest Yocto Project release
- There are different ways to overcome the shortcomings with CUDA requires by using backported GCC/G++, an external toolchain or an older Yocto releases

Thank You!

Konsulko Group

Useful links:

- https://github.com/madisongh/meta-tegra
- https://www.konsulko.com/building-a-custom-linux-distribution-for-nvidia-cuda-enabled-embedded-devices/
- https://github.com/madisongh/meta-tegra/blob/master/README
- https://github.com/madisongh/meta-tegra/wiki/Using-linaro-gcc7-for-CUDA-support
- https://github.com/madisongh/meta-tegra/wiki/Flashing-the-Jetson-Dev-Kit
- https://developer.nvidia.com/embedded/linux-tegra