Advanced Lab: IoT Development

David Reyna, Mark Hatle
Wind River Systems

Yocto Project Developer Day • San Jose • 6 March 2015
Agenda

• The Mission
• Activity 1: Preparing the Projects
• Activity 2: Presentation on Advanced Topics
• Activity 3: Preparing the Kernel Module Simulator
• Activity 4: Preparing the Application
• Activity 5: Prototyping the IoT Edge Device
• Activity 6: Extra Credit
• Q & A
• Appendix
  • Project Setup at Home
  • Parts List and ordering information
  • Resource links
Greetings, IoT Developer Hero!

Your company is about to release a new IoT product, but the engineer before you has been called away.

Your task is to finish this device. You will need to debug it, remotely update it, connect it to your home server, and demonstrate it to your boss, all in one 3 hour class!

But not to worry, we will show you the techniques you will need to be an IoT hero here and in the future!
The Product

- A *morse code edge device as part of an IoT product to support the severely handicapped*

- *This device must be able to connect with a local target, communicate with peers, and communicate with a remote server*
The Challenge

• **No edge device!**
  • You have no edge device yet, so you need to simulate the hardware with a custom device driver

• **No board!**
  • You have no board yet, so you need to develop and validate the application using QEMU

• **The Kernel Module has bugs!**
  • You will need your skills to debug the kernel module

• **The Application has bugs!**
  • You will need your skills to debug the application
Topics Covered

• **Building on the Yocto Project Beginning Class**
  • Layers, images, targets, modularity, debugging support

• **Advanced Topics**
  • PRserver, sysfs and GPIO, non-blocking I/O

• **Kernel Space**
  • Custom device drivers, parameter passing, sysfs, kernel timers, debugging techniques

• **Application Space**
  • Custom applications, sysfs access, non-blocking character I/O, application timers, socket servers and clients, debugging techniques

• **Board Bring-up**
  • Connecting an edge device to the board
Topics Not Covered (for sake of brevity)

- **Security**
  - Encryption on the line
  - Permission control at Sever
  - Permission control at Target

- **Network Configuration**
  - Firewalls
  - Proxies
  - Discovery

- **GUI Tools**
  - Eclipse
Activity One

Preparing the Projects
You will be given information on how to log into your personal virtual host.

The virtual host’s resources can be found here:
- Yocto Project: "/scratch/yocto/sources/poky"
- Downloads: "/scratch/yocto/downloads"
- Sstate-cache: "/scratch/yocto/sstate-cache"
- Sources: "/scratch/yocto/sources"

You will be using SSH to communicate with your virtual server, to reduce the overhead that graphical systems like VNC would require.
Class setup

- Your host use the network to communicate with your virtual host over three SSH connections
- You will communicate with your QEMU target using two serial connections
Build your QEMU project

• Prepare your QEMU project (for part 1)

```
$ cd /scratch/working
$ . /scratch/yocto/sources/poky/oe-init-build-env build-qemux86
$ vi conf/bblayers.conf

# add to conf/bblayers.conf to BBLAYERS:
   /scratch/yocto/sources/ypdd-adv-layer \\

$ vi conf/local.conf
# add to local.conf:
SSTATE_DIR = "/scratch/yocto/sstate-cache"
DL_DIR = "/scratch/yocto/downloads"
IMAGE_INSTALL_append = " gdbserver morseapp morsemod openssh"
EXTRA_IMAGEDEPENDS_append = " gdb-cross-i586 "
INHIBIT_PACKAGE_STRIP_pn-linux-yocto = "1"
INHIBIT_PACKAGE_STRIP_pn-morsemode = "1"

$ bitbake core-image-base
```
Advanced Topic: PR Service

• PR = Package Revision

• A package feed manager which is compatible with existing package manager applications like RPM/smart, Debian/apt and opkg.

• Attempting to maintain these values in the metadata is error prone, inaccurate and causes problems for people submitting recipes

• Example:
  • bash-3.2-r0.armv5l.rpm
  • bash-3.2-r1.armv5l.rpm
Advanced Topic: PR Service

- Insures that versions increase in a linear fashion and there are a number of version components that facilitate this, namely in order of decreasing priority PE, PV and PR (epoch, version and revision).

- Enable with this option:

```bash
# add to conf/local.conf:
PRSERV_HOST = "localhost:0"
```

- More information can be found here:
  - https://wiki.yoctoproject.org/wiki/PR_Service
Advanced Topic: GPIO and sysfs

• The "sysfs" virtual file system is a service that provides access from user space to managed kernel resources.

• The GPIO port access is such a service that the Linux kernel provides, and all of the boards in the class support this.

• The GPIO sysfs interface allows simple echo statements to expose ports and set their direction and value.

• To instantiate a GPIO pin in the file system, you simply:
  $ cd to "/sys/class/GPIO"
  $ echo $PORT > export
  $ cd gpio${PORT}
  $ echo "in" > direction (or) echo "out" > direction
  $ cat value (or) echo $VALUE > value

• For the QEMU simulator, we are going to use a custom kernel module to instantiate GPIO ports to simulate the hardware
Advanced Topic: Non-Blocking I/O

• For our application, we need to weave together asynchronous and time critical data from the GPIO hardware, the client-server socket, and the user’s keyboard commands

• The C standard I/O libraries are built around line control, which is blocking in nature. We get around this by:
  • Changing STDIN to not wait for EOL
  • Changing STDOUT to not wait for EOL
  • Fork the application into a child that blocks on getchar(), and passes received characters over a pipe to the main thread
  • A side effect is that unexpected exits can leave the console in this non-standard state, needing a manual "reset" command

• The socket interface is normally blocking, but this is easy to reconfigure

• The GPIO sysfs interface is by nature non-blocking, at least for the process that opens and thus reserves the channel
Advanced Topic: Updating without Rebooting

• While performing a full rebuild and re-burning boot media is the cleanest and safest model for updates, it is not a timely method when you need a quick and focused debug and deploy cycle.

• The easiest method to get updated content into the target is to use scp from SSH. Most of the images come with SSH already included, and for images like "core-image-base" you only need to append "openssh".

• *In this class, as a shameless shortcut you will see that we used the morseapp’s recipe to create a script that is inserted into the target file system that can be called to perform the pull scp for you. This recipe leverages the bitbake environment information to locate the desired generated content from their sometimes arcane positions in the tree.*

• You can also push the recipe’s generated packages (e.g. binary rpm files) to the target for more complex content installing.

• In this class we will try only reboot the QEMU a few times (for the kernel module’s sake).
Advanced Topic: Morse Code

• **Timing**
  • The "dit" (·) is one time unit long (1/2 second in this implementation)
  • The "dah" (--) is three time units long
  • The space between dits and dahs is one time unit long
  • The space between letters is three time units long
  • The space between words is seven time units long
  • In this class’s implementation, anything shorter that 3/4 second will be a dit, and anything longer will be a dah

• **Facts about Morse Code**
  • Professionals can process more that 40 words per minute
  • Several morse code readers have been developed for the handicapped, for example using breath control into air tubes
  • Morse code together with the telegraph network was the internet of its day, and ran empires
Activity Three

Debugging the Kernel Module
**Morsemod features**

- **Morsemod Sysfs files**
  - "key" : the virtual key input port
  - "led" : the virtual LED output port
  - "simkey" : the backdoor port

- **Simkey**
  - `echo '<' > simkey` (turn on the broadcast mode)
  - `echo '>' > simkey` (turn off the broadcast mode)
  - `echo '[' > simkey` (turn on the loopback mode)
  - `echo ']' > simkey` (turn off the loopback mode)
  - `cat simkey` returns the inner state: "K=0 L=1 B=1 LP=0"

- **The "loopback" will echo the Key value to the LED value**
- **The "broadcast" mode will repeat a pre-recorded morse message**
Debugging the Kernel Module

• We are now going to use KGDB to test and debug the kernel module

• We will show that when encountering an error, it is important to always check dmesg first before going to a debugger

• We will be using KGDBoS to debug the kernel over the serial port

• We will in fact need to set up QEMU with two serial ports, (a) one for the user console and (b) one for the secondary port for the KGDB access

• You should have three (3) windows open: build, target console and remote GDB.

• QEMU has a very powerful backend JTAG-like debugging facility; we will skip that in this demonstration to show techniques that would carry over to a physical target
Interlude – Magic Numbers

• When you assign serial ports in QEMU, they are automatically mapped to ttyS0, ttyS1, and so forth

```bash
$ runqemu qemux86 nographic \ qemuparams="-serial telnet:localhost:2345,server \ -serial tcp:localhost:2346,server,nowait"
```

• That is why when we start the KGDBoC service we will specify ttyS1, because your telnet session already took ttyS0

```bash
$ telnet localhost 2345
# echo ttyS1 > /sys/module/kgdboc/parameters/kgdboc
```

• Note that different arches sometimes have different port names, For example, qemuarm uses ttyAMA0. You may need to consult the device tree to see what was instantiated.
Debugging with KGDB – error inserting module

- Build Window: Start the QEMU session with two serial ports

```bash
$ runqemu qemux86 nographic \
qemuparams="-serial telnet:localhost:2345,server \ 
-serial tcp:localhost:2346,server,nowait"
```

(Note: to stop QEMU, ctrl-c in the Build Window)

- Console Window: Start a user console, insert the module, and it fails!

```bash
$ telnet localhost 2345
qemux86 login: root
root@qemux86:~# modprobe morsemod
<Error!>
```

- What happened? First look at dmesg or scrollback!

```bash
# dmesg tail
...
52 sysfs_warn_dup+0x7a/0x90()
sysfs: cannot create duplicate filename '/kernel/morse-mod/key'
Modules linked in: morse_mod(0+) uvesafb...
```
Debugging the kernel module – fix #1

• Bug -- we tried to create the 'key' file twice... whoops!
  *(trying to use the debugger on this will end up taking twice as long or worse..)*

• Let’s debug this:

  $ cd .../ypdd-adv-layer/recipes-ypdd-adv/morsemod/morsemod-2014.10.0
  $ vi morsemod.c

• Look for __ATTR (the structure that holds the sysfs file info)

```c
static struct kobj_attribute simkey_attribute =
  __ATTR(key, 0666, simkey_show, simkey_store);
...
static struct kobj_attribute led_attribute =
  __ATTR(led, 0666, b_show, b_store);
static struct kobj_attribute key_attribute =
  __ATTR(key, 0666, b_show, b_store);
```

• Whoops, we defined 'key' twice, top one should be 'simkey'!

```
- __ATTR(key, 0666, simkey_show, simkey_store);
+ __ATTR(simkey, 0666, simkey_show, simkey_store);
```
Debug the kernel module – try again

• Build Window: Boot, try it again (see above)

```bash
$ bitbake core-image-base
$ runqemu qemux86 nographic qemuparams="-serial \n
telnet:localhost:2345,server \n
-serial tcp:localhost:2346,server,nowait"
```

• Console Window: load module and see if it works...

```bash
$ telnet localhost 2345
qemux86 login: root
root@qemux86:~# modprobe morsemod
root@qemux86:~# cd /sys/kernel/morsemod
root@qemux86:/sys/kernel/morsemod# cat simkey
K=0 L=0 B=0 LP=0
```
Debug the kernel module – test loopback

- Let’s see if it can remember the key and led values

```
root@qemux86:/sys/kernel/morsemod# echo '1' > key
root@qemux86:/sys/kernel/morsemod# cat simkey
K=1 L=0 B=0 LP=0
root@qemux86:/sys/kernel/morsemod# echo '0' > key
root@qemux86:/sys/kernel/morsemod# cat simkey
K=0 L=0 B=0 LP=0
root@qemux86:/sys/kernel/morsemod# echo '1' > led
root@qemux86:/sys/kernel/morsemod# cat simkey
K=0 L=1 B=0 LP=0
root@qemux86:/sys/kernel/morsemod# echo '0' > led
root@qemux86:/sys/kernel/morsemod# cat simkey
K=0 L=0 B=0 LP=0
```

- Let’s see if we can enable loopback...

```
root@qemux86:/sys/kernel/morsemod# echo '[' > simkey
root@qemux86:/sys/kernel/morsemod# cat simkey
K=0 L=0 B=0 LP=1
root@qemux86:/sys/kernel/morsemod# echo '1' > key
root@qemux86:/sys/kernel/morsemod# cat simkey
K=1 L=1 B=0 LP=1
root@qemux86:/sys/kernel/morsemod# echo '0' > key
root@qemux86:/sys/kernel/morsemod# cat simkey
K=0 L=0 B=0 LP=1
root@qemux86:/sys/kernel/morsemod# echo ']' > simkey
root@qemux86:/sys/kernel/morsemod# cat simkey
K=0 L=0 B=0 LP=0
```
Debug the kernel module – Bug #2 : broadcast

• Let’s try enabling the demo broadcast message...

```
root@qemux86:/sys/kernel/morsemmod# echo '<' > simkey
root@qemux86:/sys/kernel/morsemmod# cat simkey
K=0 L=0 B=1 LP=0
```

• It doesn't seem to be working. Time to try to debug the module!
Debug the kernel module – Bug #2: broadcast

- Production image usually lack debug capabilities (stripped symbols, no included debug sources, etc.) In the Yocto Project, debug information is separate from the runtime in most cases. It is packaged into ‘-dbg’ labeled packages.

- By creating a companion debug capable filesystem, you can use remote (cross) debugging to debug a production filesystem.

- **Debug Window: Construct a debug-filesystem**
  - Extract ‘dbg’ packages (rpm in this example), user space and kernel
  - Add the production filesystem components

```bash
$ rm -rf debugfs ; mkdir debugfs ; cd debugfs
$ for pkg in ../tmp/deploy/rpm/*/*-dbg* ; do
    ../tmp/sysroots/x86_64-linux/usr/lib/rpm/rpm2cpio $pkg | cpio -i ; done
$ ../tmp/sysroots/x86_64-linux/usr/lib/rpm/rpm2cpio
    ../tmp/deploy/rpm/*/kernel-dev* | cpio -i
$ tar xvfj ../tmp/deploy/images/qemuarm/core-image-base-qemuarm.tar.bz2
$ cd ..
```
Magic Numbers

• The script that gathers the user space debug RPM files assumes that you have only one arch.

$ for pkg in ../tmp/deploy/rpm/*/*-dbg* ; do
    ../tmp/sysroots/x86_64-linux/usr/lib/rpm/rpm2cpio $pkg | cpio -i ; done

• If you can built multiple archs, then you need to explicitly select the ones you need else you may end up with the wrong content.

• For this example where you have both qemu86 and qemuarm built, for qemu86 you will only want the RPMs from "all", "i586", and "qemu86".

$ ls tmp/deploy/rpm/
all armv5te i586 qemuarm qemu86
$
Find the Kernel Source Substitution

- **We need to derive the path to the kernel source**
  - This is a little tricky, in that the path is subject to change since it contains build stamp information. The easiest way of doing this is simply to inspect the vmlinux on the host.
  - Do an objdump, and search for a file directory entry (there should be some near the top). These are what GDB uses to look up the paths for the corresponding file entries.

```bash
$ objdump --dwarf boot/vmlinux-3.14.26ltsi-yocto-standard | less
```

The Directory Table:
```
<projdir>/tmp/work/qemux86-poky-linux-gnueabi/linux-yocto/3.14.24+gitAUTOINC+a2…47-r0/linux/arch/arm/include/asm
<projdir>/tmp/work/qemux86-poky-linux-gnueabi/linux-yocto/3.14.24+gitAUTOINC+a2…47-r0/linux/arch/arm/kernel
```

- The common ancestor in the example above is:

  ```
  <projdir>/tmp/work/qemux86-poky-linux-gnueabi/linux-yocto/3.14.24+gitAUTOINC+a2…47-r0/linux
  ```

- **Remember this path for the GDB substitution step next**
  - *Note: if you do not issue the substitute-path, the system will attempt to resolve on the local machine and the path specified. This -will- work in many cases, as the user will still have their project available. But where they don't, it will fail.*
KGDBoC - Start

• Console Window: enable KGDBoC

```bash
# echo ttyS1 > /sys/module/kgdboc/parameters/kgdboc
# echo g > /proc/sysrq-trigger
(system will stall and drop into KDB on the ‘ttyS1’ serial port)
```

(See: https://www.kernel.org/doc/htmldocs/kgdb/EnableKGDB.html)

• Debug Window: Connect to the remote kernel (KGDBoS)
  • Use kernel source substitution path from previous page

```bash
$ ./tmp/sysroots/x86_64-linux/usr/bin/i586-poky-linux/i586-poky-linux-gdb
(gdb) set sysroot debugfs
(gdb) set substitute-path /usr/src/debug debugfs/usr/src/debug
(gdb) set substitute-path <projdir>/tmp/work/qemux86-poky-linux-gnueabi/linux-yocto/3.14.24+gitAUTOINC+a2...47-r0/linux debugfs/usr/src/kernel
Reading symbols from /scratch/working/build-qemuarm/debugfs/boot/vmlinux-3.14.26ltsi-yocto-standard...done.
(gdb) target remote localhost:2346
Remote debugging using localhost:2346
kgdb_breakpoint ()
at /scratch/working/build-qemuarm/tmp/work/qemux86-poky-linux/linux-yocto/...1043 wmb(); /* Sync point after breakpoint */
(gdb)
```
KGDBoC - Connect

- Debug Window: Load into GDB the kernel module

```gdb
(gdb) monitor lsmod
Module       Size  modstruct    Used by
morsemod    4896  0xbf05a778   0     (Live) 0xbf05a000 [ ]
nfsd        292794 0xbf0395b0   11    (Live) 0xbf000000 [ ]
```

```gdb
(gdb) add-symbol-file \ debugfs/lib/modules/3.14.26lti-yocto-standard/extra/morsemmod.ko 0xbf05a000
add symbol table from file "debugfs/lib/modules/3.14.26lti-yocto-standard/extra/morsemmod.ko" at .text_addr = 0xd081a000
(y or n) y
Reading symbols from /scratch/working/build-qemuarm/debugfs/lib/modules/3.14.26lti-yocto-standard/extra/\ morsemmod.ko...(no debugging symbols found)...done.
(gdb)
```

- When broadcast mode is enabled what we're trying to do is iterate over the 'message_str' value using the message_index, so set the break point at the top of the: "if ( broadcast == 1 )"

```gdb
(gdb) break morsemod.c:68
(gdb) c
```
KGDBoC - walking

- Console Window:

```bash
root@qemux86:~# cd /sys/kernel/morsemod
root@qemux86:/sys/kernel/morsemod# echo '<' > simkey
```

- Debug Window:

Use 'n' to walk through the code flow

NOTE: You likely can't inspect variables (might be a bug in gdb?)

But if you disassemble you'll see something like:
(disassemble /m)

```
73  message_index++;
    0xd081a0ea <+90>:    add    $0x1,%ebx
    0xd081a0f7 <+103>:  mov     %ebx,0xd081a7c0
```

So the value is in the location pointed to by 0xd081a7c0:
(gdb) print *(int*)0xd081a7c0

Eventually you see:
```
74  if (message_index <= strlen(message_str)) {
(gdb)
75  message_index=0;
```

Wait, the message_index keeps getting set back to 0! We should be checking if it -overflows-, not -underflows-!
KGDBoC – fix bug #2

• Change the code to be

```c
-   if (message_index <= strlen(message_str)) {
+   if (message_index >= strlen(message_str)) {
```

• Rebuild ...
  • (refer to previous steps for build and boot -- remember to regen the debugfs, slides 30-32)

• Restart the module ...

• Console Window:

```
telnet localhost 2345
qemux86 login: root
root@qemux86:~# modprobe morsemmod
root@qemux86:~# cd /sys/kernel/morsemmod
root@qemux86:/sys/kernel/morsemmod# cat simkey
K=0 L=0 B=0 LP=0
root@qemux86:/sys/kernel/morsemmod# echo '<' > simkey
root@qemux86:/sys/kernel/morsemmod# cat simkey
K=1 L=0 B=1 LP=0
root@qemux86:/sys/kernel/morsemmod# while true ; do cat key ; sleep 1 ; done
```
KGDBoC – Bug #3 : stopping broadcast

- Does it work yet...

```bash
root@qemux86:/sys/kernel/morsemod# while true ; do cat key ; sleep 1 ; done
1
1
0
1
1
1
.
.
.
```

- Great, it works! Now turn it off...

```bash
.
^C
root@qemux86:/sys/kernel/morsemod# echo '>' > simkey
root@qemux86:/sys/kernel/morsemod# cat simkey
K=0 L=0 B=1 LP=0
root@qemux86:/sys/kernel/morsemod# echo '>' > simkey
root@qemux86:/sys/kernel/morsemod# cat simkey
K=0 L=0 B=1 LP=0
```

- Cannot turn it off, new bug
KGDBoC - Reconnect

• Console Window: Setup kgdb/gdb again ...

```bash
echo ttyS1 > /sys/module/kgdboc/parameters/kgdboc
echo g > /proc/sysrq-trigger
```

• Debug Window

```
$ ./tmp/sysroots/x86_64-linux/usr/bin/i586-poky-linux/i586-poky-linux-gdb
(gdb) set sysroot debugfs
(gdb) set substitute-path /usr/src/debug debugfs/usr/src/debug
(gdb) set substitute-path <projdir>/tmp/work/qemux86-poky-linux-gnueabi/linux-yocto/3.14.24+gitAUTOINC+a2...47-r0/linux debugfs/usr/src/kernel
Reading symbols from /scratch/working/build-qemuarm/debugfs/boot/vmlinux-3.14.26ltsi-yocto-standard...done.
Reading symbols from `..`
(gdb) target remote localhost:2346
Remote debugging using localhost:2346
kgdb_breakpoint ()
at /scratch/working/build-qemuarm/tmp/work/qemux86-poky-linux/linux-yocto/...
1043 wmb(); /* Sync point after breakpoint */
(gdb)
```
KGDBoC – break in 'simkey_store'

- ... and monitor the module ...

```
(gdb) monitor lsmod
Module      Size  modstruct  Used by
morsemmod  3973   0xd081a640 0 (Live) 0xd081a000 [ ]
nfsd       206660 0xd18f1160 11 (Live) 0xd18c6000 [ ]
ivesafb    22894  0xd083baa0 1 (Live) 0xd0837000 [ ]
(gdb) add-symbol-file
debugfs/lib/modules/3.14.26ltsi-yocto-standard/extra/morsemmod.ko 0xd081a000
add symbol table from file
   .text_addr = 0xd081a000
(y or n) y
Reading symbols from
/scratch/working/build-qemuarm/debugfs/lib/modules/.../morsemmod.ko...done.
(gdb)
```

- We know the bug is likely in 'simkey_store' because that is the routine that is called when the user attempt to write to the 'simkey' file to control behaviors

```
(gdb) break simkey_store
Breakpoint 1 at 0xd081a000: file
/scratch/working/build-qemuarm/tmp/work/qemu86-poky-linux/.../morsemmod.c,
   line 193.
(gdb) c
```
KGDBoC – stepping to bug
• Does it work yet... Console Window:

```bash
echo '>' > simkey
```

• Debug Window:

```
(gdb) list
(centered on the first line of the function...)

verify the input was processed correctly

(gdb) print buf[0]
$1 = 62 '>'
```

• Step through the code to where '>' is processed.
• Observe that due to optimization it skips over that line! Something must be wrong here!
KGDBoC - disassembly

- If you disassemble the code (disassemble /m), you'll see that both '<' and '>' are setting '1'

```
196 if ('<' == buf[0]) broadcast=1;
    0xd081a008 <+8>:  movzbl (%ecx),%eax
    0xd081a00b <+11>:  mov %eax,%edx
    0xd081a00d <+13>:  and $0xffffffff,%edx
    0xd081a010 <+16>:  cmp $0x3c,%dl
    0xd081a013 <+19>:  je 0xd081a040 <simkey_store+64>
    0xd081a015 <+21>:  cmp $0x5b,%al
    0xd081a017 <+23>:  je 0xd081a060 <simkey_store+96>
    0xd081a063 <+99>:  movl $0x1,0xd081a7c8

    ...pointed to by 0xd081a7c4

(gdb) print *(int *)0xd081a7c4
$2 = 1
```
**KGDBoC – manually changing variables**

- We can manually clear the value (note this has to be done *after* the code sets it to 0)

```c
(gdb)
kobj_attr_store (kobj=<optimized out>, attr=<optimized out>,
    buf=<optimized out>, count=2)
at
/scratch/working/build-qemuarm/tmp/work/qemux86-poky-linux/.../kobject.c:776
776     }   
(gdb) set *(int *)0xd081a7c4 = 0
(gdb) print *(int *)0xd081a7c4
$3 = 0
(gdb) c
```

- Console Window:

```
(root@qemux86:/sys/kernel/morsemod# cat simkey
K=0 L=0 B=0 LP=0
```

- OK we found the bug, go fix the code, retest the code

```c
-   else if ('>' == buf[0]) broadcast=1;
+   else if ('>' == buf[0]) broadcast=0;
```

- Observe that it now passes all of the tests!
Activity Four

Preparing the Application
Exercising the morseapp Application

• We will exercise the morseapp features one step at a time
• This will allow us to verify each feature as we build up to the full functionality
• This will also provide you with the templates to easily fork and build your own new features
• These are the morseapp components:
  • morse_app: set up the non-blocking I/O, configure the interface, provide the basic talk modes
  • morse_gpio: manage the GPIO sysfs interface
  • morse_codec: manage the morse/text translation
  • morse_server: manage the morseapp local/remote server
  • morse_client: manage the morseapp client
Run the morseapp Application

- Start morseapp first time
  
  ```
  # /usr/bin/morseapp
  ```

- Or re-fetch and start morseapp after a rebuild
  
  ```
  # /opt/upload_morseapp.sh
  <enter your host password>
  # ./morseapp
  ```

- How? The shameless hack in bb file! Bitbake knows where the content is, so you can let it do the work. For the sake of this class’s setup, you will need to insert your special port number into the scp command using "-P portnum"

```bash
# /usr/bin/morseapp

do_install() {
  ...
  # shameless hacking: add helper upload scripts
  mkdir -p ${D}/opt
  export ipaddr=`ifconfig eth0 | grep "inet addr" | sed -e "s/.*inet addr://" -e "s/ .*/"`
  echo "scp -P portnum $USER@$ipaddr:${D}/usr/bin/morseapp ." > ${D}/opt/upload_morseapp.sh
  echo "scp -P portnum $USER@$ipaddr:`ls ${TOPDIR}/tmp/work/*/morsemodule/0.1-r0/morsemodule.ko` "." > ${D}/opt/upload_morsemodule.sh
  echo "insmod morsemodule.ko ; echo 8 > /sys/kernel/morsemodule/simkey" > ${D}/opt/broadcast_morsemodule.sh
  chmod +x ${D}/opt/upload_morseapp.sh
  chmod +x ${D}/opt/upload_morsemodule.sh
  chmod +x ${D}/opt/broadcast_morsemodule.sh
}
```
Step by Step: the morseapp Application (1)

- Set the input device as the simulator with commands "1,5,m"

- The "loopback" mode will echo changed values from the morsemod 'device'

  [ Loopback : device keys are echoed back to the device LED ]
  Type '#' to quit

  Device Key: [*]

- The "talk morse code" mode will support a morse conversation between the app and the ‘device’, with backdoor keys for morsemod

  [ Talk Morse code ]

  Type the '.' period key to toggle your 'key'
  * a 'dit' is about 1/2 seconds
  * a 'dah' is about 1 1/2 seconds (three 'dits')
  * a letter space is about 1/2 seconds
  * a word space is about 1 1/2 seconds

  Type these keys for the simulated device
  '/' : toggle the device's key
  '<' : broadcast mode on
  '>' : broadcast mode off

  Type '#' to quit

  UserKey: [*] | DeviceKey: [*] || SIM: KEY LED
Step by Step: the morseapp Application (2)

- The "talk text" mode will support a text-over-morse conversation between the app and the ‘device’

```
[ Talk Text (ASCII) ]
Type your text message:
* use the letters a-z, 0-9 (case is ignored)
  * use a space for word separation
Type these keys for the simulated device
  '<' : broadcast mode on
  '>' : broadcast mode off
  '\': force clear the out buffer
Type '#' to quit

In: [????___________________] (** ) | Out: [____________________] ()
```

- Hit the "<" to remotely start the broadcast mode

- Observe that there are dits but no dahs coming in and that cannot be right - time to bring out the debugger!

- Quit the application with "#" and ‘q’
Using GDB - Setup

- Let’s step into the program and see how to configure the debugging

- Console Window:

  ```
  root@qemux86:~# gdbserver /dev/ttyS1 morseapp
  Process morseapp created; pid = 458
  Remote debugging using /dev/ttyS1
  ```

- Debug Window:

  ```
  ./tmp/sysroots/x86_64-linux/usr/bin/i586-poky-linux/i586-poky-linux-gdb
  
  (gdb) set sysroot debugfs
  (gdb) set substitute-path /usr/src/debug debugfs/usr/src/debug
  (gdb) file debugfs/usr/bin/morseapp
  Reading symbols from /scratch/working/build-qemuarm/debugfs/usr/bin/morseapp...Reading symbols from /scratch/working/build-qemuarm/debugfs/usr/bin/.debug/morseapp...done.
  done.
  (gdb) target remote localhost:2346
  Remote debugging using localhost:2346
  Reading symbols from debugfs/lib/ld-linux.so.2...Reading symbols from /scratch/working/build-qemuarm/debugfs/lib/ld-linux.so.2...done.
  Loaded symbols for debugfs/lib/ld-linux.so.2
  0x4d872d00 in _start () from debugfs/lib/ld-linux.so.2
  (gdb)
  ```
Using GDB - running

• The application was started in the halted position, so let’s let it run *(note if you see references or failed to load messages for linux-gate.so.1, that's a virtual library that the kernel sets up for syscalls - there is nothing to load)*

• Debug Window:
  
  ```
  (gdb) c
  ```

• In the console follow the steps to enable Talk Text w/ Broadcast (1,5,m,4,>)
  
  • Use (ctrl-c) to signal the application and debugger with a SIGINT

• Program received signal SIGINT, Interrupt.
  0x4d9381fb in __nanosleep_nocancel () at ../sysdeps/unix/syscall-template.S:81
  81      T_PSEUDO (SYSCALL_SYMBOL, SYSCALL_NAME, SYSCALL_NARGS)
  (gdb) bt
  #0 0x4d9381fb in __nanosleep_nocancel ()
      at ../sysdeps/unix/syscall-template.S:81
  #1 0x4d968273 in usleep (useconds=useconds@entry=50000)
      at ../sysdeps/unix/sysv/linux/usleep.c:32
  #2 0x080494ec in talk_text () at morse_app.c:263
  #3 0x08049de9 in parent (p=p@entry=0xbfffffd18, child_pid=460)
      at morse_app.c:524
  #4 0x08048d68 in main () at morse_app.c:457
Using GDB - stepping with ‘finishing’

- Lets take a look at `talk_text` (*use 'finish' to complete the current function and return to the caller*)

```
(gdb) finish
Run till exit from #0 0x4d9381fb in __nanosleep_nocancel ()
   at ../sysdeps/unix/syscall-template.S:81
usleep (useconds=useconds@entry=50000)
   at ../sysdeps/unix/syv/linux/usleep.c:33
33    }
(gdb) finish
Run till exit from #0 usleep (useconds=useconds@entry=50000)
   at ../sysdeps/unix/syv/linux/usleep.c:33
0x080494ec in talk_text () at morse_app.c:263
263    usleep(50000);  
Value returned is $1 = 0
(gdb) finish
Run till exit from #0 0x4d9381fb in __nanosleep_nocancel ()
   at ../sysdeps/unix/syscall-template.S:81
usleep (useconds=useconds@entry=50000)
   at ../sysdeps/unix/syv/linux/usleep.c:33
33    }
(gdb) finish
Run till exit from #0 usleep (useconds=useconds@entry=50000)
   at ../sysdeps/unix/syv/linux/usleep.c:33
0x080494ec in talk_text () at morse_app.c:263
263    usleep(50000);  
Value returned is $1 = 0
```
Using GDB - stepping with ‘next’

- **We can step the program with 'next'**
  - Note since the code is optimized stepping is out of order!
  - We can also use 'until' to step until the next line of code.

```
(gdb) n
263   usleep(50000);
(gdb) until
227       strcpy(disp_outTextStr,outTextStr);
```

*Also using 'bt' to get a look at the function back trace:*

```
(gdb) bt
#0  talk_text () at morse_app.c:227
#1  0x08049de9 in parent (p=p@entry=0xbffffd18, child_pid=460)
    at morse_app.c:524
#2  0x08048d68 in main () at morse_app.c:457
```

- **Lets set a break point on 'scan_morse_in()'**

```
(gdb) break scan_morse_in
Breakpoint 1 at 0x8049f90: file morse_codec.c, line 145.
```
Using GDB - continuing to the bug

- Continue execution until we hit that function

```c
(gdb) c
Continuing.

Breakpoint 1, scan_morse_in () at morse_codec.c:145
145   char scan_morse_in(void) {

(list the sources - repeat 'list' until you see the below code)
(gdb) list

(gdb) break 159
(gdb) clear scan_morse_in
(gdb) c
Breakpoint 2, scan_morse_in () at morse_codec.c:159
159     strcat(inMorseBuf,"*"');

(gdb) list
154     lowcnt++;
155
156     /* parse the 1->0 transition */
157     if ((1 == lastkey) && (0 == key)) {
158         if (highcnt <= dit_max_cnt)
159             strcat(inMorseBuf,"*"');
160         else
161             strcat(inMorseBuf,"*"');
162     highcnt=0;
163 }
```

- You can see around 159 and 161 where the * and - are put into the buffer... lets break on the *...
Using GDB – finding the bug

- Ok, the problem is that we are returning dits for both short and long signals, and the fix is simple

```c
157     if ((1 == lastkey) && (0 == key)) {
158         if (highcnt <= dit_max_cnt)
159             strcat(inMorseBuf,"*");
160     } else
- 161         strcat(inMorseBuf,"*");
+ 161         strcat(inMorseBuf,"-" bons
162         highcnt=0;
163     }
```

- You can now fix and rebuild
Using GDB - extra credit

• BTW, we can inspect the variables…

(gdb) print lastkey
$6 = 1 '\001'
(gdb) print key
$7 = <optimized out>
(gdb) print highcnt
$8 = 30

• Note the "optimized out", tricks like what we did in the kernel may be needed here... otherwise we can watch the code flow and determine the likely values!

• Note if you are ever looking at a file and are not sure where the source is, use ‘info source’

(gdb) info source
Current source file is morse_codec.c
Compilation directory is /usr/src/debug/morseapp/2014.10.0-r0
Located in
/scratch/working/build-qemuarm/debugfs/usr/src/debug/morseapp/2014.10.0-r0/morse_codec.c
Contains 247 lines.
Source language is c.
Compiled with DWARF 2 debugging format.
Does not include preprocessor macro info.
Using GDB - extra credit

• You can do the same debugging process as above, but instead of using a serial console, use TCP/IP if your device supports it.

• Simply replace the /dev/ttyS1 with the address and port of the GDB client.

# Console

root@qemux86:~# gdbserver 192.168.7.1:2345 morseapp
Process morseapp created; pid = 456
Listening on port 2345

# Debug - Server connect such as:

target remote 192.168.7.2:2345
Step by Step: the morseapp Application (3)

- The working "talk text" mode will support a text-over-morse conversation between the app and the 'device'

  [ Talk Text (ASCII) ]
  Type your text message:
  * use the letters a-z, 0-9 (case is ignored)
  * use a space for word separation
  Type these keys for the simulated device
  '<' : broadcast mode on
  '>' : broadcast mode off
  '\' : force clear the out buffer
  Type '#' to quit

  In: [?p dev_____________] (* ) | Out: [adbqwer_____________] (--- *--- *+4)

- The "talk local server" mode will support simple morse code queries to a local server

  [ Client to Local Server Mode : respond to device queries ]
  Server commands are:
  'e' ('*')   : echo an 'e' ('*')
  'd' ('-**'): echo the day of the week ('mon' .. 'sun')
  't' ('-')   : echo the time ('hhmms')
  'f' ('**-*'): echo a fortune place
  * use a space for word separation
  Type these keys for the simulated device:
  '/' : toggle the device's key
  '\' : force clear the out buffer
  Server prompt at device is 's' ('---')
  Type '#' to quit

  In: [________?____] (- ) | Out: [_______________] ()
  Local server:d => 'Mon'
  In: [________?____] (---- ) | Out: [_______mon s_______] (--- -* +5)
Step by Step: the morseapp Application (4)

• Test the server/client functionality
  
  Setup the Server/Client test connection, using the default address "localhost"
  
  • In one shell, start the test server with the "s" command
  • In a second shell, start the test client with the "t" command
  • Observe the passing of test strings between these components

• Run the full server/client functionality
  
  Setup the Server/Client test connection, using the default address "localhost"
  
  • In one shell, start the morseapp server with the "7" command
  • In a second shell, start the client with the "6" command
  • Use the "/" key to insert morse code clicks, observe the server replies
  • It should work like the "local server" but this time over a local socket
Step by Step: the morseapp Application (5)

- Connect to the class remote server
  - Set the class’s remote server address in the configuration page ("1" > "7") and start the client again
  - It should work like the "local server" but this time over a remote socket
Activity Five

Prototyping the IoT Edge Device
Building a Prototype Edge Device

- This next section describes how you can build your own prototype device, using a breadboard and a target’s GPIO pins

- There are three target examples included:
  - Beaglebone Black
  - Minnowboard MAX
  - Wand Board Quad
Notes about GPIO (1)

- Double check that the GPIO pins you wish to use are actually available (e.g. BeagleBone GPIO_20), Here is the test:
  
  ```
  $ cd /sys/class/gpio
  $ export pin=20
  $ echo $pin > export
  $ cd gpio${pin}
  $ echo out > direction
  $ echo 1 > value
  $ cat value
  
  If the value does not equal 1, then the pin is not fully enabled
  ```

- Make sure that the voltages applied match the accepted range, as most boards use 3.3v for their pins.
  - The WandBoard has only +5v available on the header yet requires 3.3v for its inputs. You can use a voltage regulator from the 5 volts, but in this case since we are using low current we can simply use a voltage divider resistor network to get the +3.3v reference.
Notes about GPIO (2)

• Be aware that some GPIO pins can only drive or pull a low amount of current
  • The Minnowboard Max is an example. In this case use can use a simple FJN3302R Transistor (with bias resistors 10K,10K) to provide the extra pull down current we need for the LED.

• The extension modules from the board vendors can be your inspiration for your own circuits
  • Most board vendors in this space publish the detailed schematics of both the boards and the many available I/O devices that they provide, which can provide examples on how exactly to properly design your own interfaces
BeagleBone Black: GPIO Layout

Note: GPIO_20 at least is not actually free
Schematic: Beaglebone Black

P9-4 +3.3 V

P9-42 LED

P9-12 Key

P9-46 GND

220 ohm=red,red,brown  4.7K=yellow,violet,orange
Sample Breadboard Layout: Beaglebone Black

P9-46 GND
P9-4 +3.3 V
P9-42 LED
P9-12 Key

220 ohm=red,red,brown  4.7K=yellow,violet,orange

4.7K Ohm
220 Ohm
Minnowboard Max: GPIO Layout

- [http://www.elinux.org/Minnowboard:MinnowMax#Low_Speed_Expansion_Top_28Top_29](http://www.elinux.org/Minnowboard:MinnowMax#Low_Speed_Expansion_Top_28Top_29)

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Pin</th>
<th>Linux GPIO #</th>
<th>Linux GPIO #</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ground</td>
<td>Gnd</td>
<td>1</td>
<td>210</td>
<td>225</td>
<td>Gnd</td>
<td>+3V3</td>
</tr>
<tr>
<td>+5V Power</td>
<td>VCC</td>
<td>3</td>
<td>221</td>
<td>224</td>
<td>GpioUART1_TXD</td>
<td>UART Transmit</td>
</tr>
<tr>
<td>SPI Chip Select 1</td>
<td>GPIO_SPI_CS#</td>
<td>5</td>
<td>220</td>
<td>225</td>
<td>GpioUART1_RXD</td>
<td>UART Receive</td>
</tr>
<tr>
<td>Master In / Slave Out</td>
<td>GPIO_SPI_MISO</td>
<td>7</td>
<td>221</td>
<td>224</td>
<td>GpioUART1_CTS</td>
<td>CTS / GPIO</td>
</tr>
<tr>
<td>Master Out / Slave In</td>
<td>GPIO_SPI_MOSI</td>
<td>9</td>
<td>222</td>
<td>227</td>
<td>GpioUART1_RTS</td>
<td>RTS / GPIO</td>
</tr>
<tr>
<td>SPI Clock</td>
<td>GPIO_SPI_CLK</td>
<td>11</td>
<td>223</td>
<td>226</td>
<td>GpioUART1_RTS</td>
<td></td>
</tr>
<tr>
<td>Clock / GPIO</td>
<td>GPIO_I2C_SCL (I2C #5)</td>
<td>13</td>
<td>243</td>
<td>216</td>
<td>GpioI2C_CLK</td>
<td>Clock / GPIO</td>
</tr>
<tr>
<td>Data / GPIO</td>
<td>GPIO_I2C_SDA (I2C #5)</td>
<td>15</td>
<td>242</td>
<td>217</td>
<td>GpioI2C_FRM</td>
<td>Frame / GPIO</td>
</tr>
<tr>
<td>UART Transmit / GPIO</td>
<td>GPIO_UART2_TXD</td>
<td>17</td>
<td>229</td>
<td>219</td>
<td>GpioI2S_DO</td>
<td>Data Out / GPIO</td>
</tr>
<tr>
<td>UART Receive / GPIO</td>
<td>GPIO_UART2_RXD</td>
<td>19</td>
<td>228</td>
<td>218</td>
<td>GpioI2S_DO</td>
<td>Data In / GPIO</td>
</tr>
<tr>
<td>GPIO / Wakeup</td>
<td>LED-- GPIO_SS_0</td>
<td>21</td>
<td>82</td>
<td>248</td>
<td>Gpio_PWM0</td>
<td>PWM / GPIO</td>
</tr>
<tr>
<td>GPIO / Wakeup</td>
<td>GPIO_SS_1</td>
<td>23</td>
<td>83</td>
<td>249</td>
<td>Gpio_PWM1</td>
<td>PWM / GPIO</td>
</tr>
<tr>
<td>GPIO / Wakeup</td>
<td>KEY-- GPIO_SS_4</td>
<td>25</td>
<td>84</td>
<td>208</td>
<td>GPIO_JBL_8254</td>
<td>Timer / GPIO</td>
</tr>
</tbody>
</table>

Note: The pins 21, 23, and 25 are the free GPIO pins.
Schematic: Minnowboard Max

JP1-4 +3.3V

JP1-21 LED

JP1-25 Key

JP1-2 GND

220 ohm = red, red, brown  4.7K = yellow, violet, orange

220 ohm

4.7K

FJN3302R Transistor With bias Resistors (10K, 10K)
Sample Breadboard Layout: Minnowboard Max

JP1-2: GND
JP1-4: +3.3 V
JP1-21: LED
JP1-25: Key

220 ohm = red, red, brown
4.7K ohm = yellow, violet, orange
WandBoard: GPIO Layout


<table>
<thead>
<tr>
<th>iMX6 GPIO (bank, number)</th>
<th>sysfs name</th>
<th>JP-4 pin</th>
<th>MorseApp</th>
</tr>
</thead>
<tbody>
<tr>
<td>+5 V</td>
<td></td>
<td>2</td>
<td>+5 V</td>
</tr>
<tr>
<td>GPIO(3, 11)</td>
<td>gpio75</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>GPIO(3, 27)</td>
<td>gpio91</td>
<td>6</td>
<td>LED</td>
</tr>
<tr>
<td>GPIO(3, 26)</td>
<td>gpio90</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>GPIO(6, 31)</td>
<td>gpio191</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>GPIO(3, 8)</td>
<td>gpio72</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>GPIO(1, 24)</td>
<td>gpio24</td>
<td>10</td>
<td>KEY</td>
</tr>
<tr>
<td>GPIO(4, 5)</td>
<td>gpio101</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>GPIO(7, 8)</td>
<td>gpio200</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td></td>
<td>20</td>
<td>GND</td>
</tr>
</tbody>
</table>

Note: use 20 gauge solid core wire for temporary jumping, but soldering to a male-male 0.05 header that you plug in is the recommend connection method for breadboarding.
Schematic: WandBoard Quad

H4-2  +5 V

H4-6  LED

H4-10 Key

H4-20 GND

220 ohm=red,red,brown  4.7K=yellow,violet,orange  1.2K=brown,red,orange  2.4K=red,yellow,orange
Sample Breadboard Layout: Wandboard Quad

H4-20
GND

H4-2
+5 V

H4-6
LED

H4-10
Key

Jumper

2.4K Ohm

1.2K Ohm

4.7K Ohm

4.7K Ohm

(+3.3v)

220 Ohm

220 ohm=red,red,brown 4.7K=yellow,violet,orange 1.2K=brown,red,orange 2.4K=red,yellow,orange
Activity Eight

• Extra Credit
Extra Credit Exercises - 1

• Add punctuation
  • There are morse codes for many punctuation marks. Add them to the application’s codec (see for example "wikipedia").

•Changing the ‘dit’ time reference
  • Add code to the kernel module to allow user configuration of the broadcast mode timing (e.g. make it go faster)?
  • Add code to the application to also allow configuration of the morse code timing translation, and also pass that setting to the kernel module?

• Pushing updates instead of pulling
  • Modify the morseapp recipe to generate a script that pushes instead of pulls updated content to the target. Note that while the default root password makes it easier, each restart of QEMU results in a different SSH signature.
Extra Credit Exercises - 2

• Setting the timers to account for delays
  • The timers in the morseapp scan the keypad input port every 1/20 second, but what the code is really doing is adding a 1/20 delay after all the loop’s code (like printf and socket I/O)

  How would you re-write the code such that it accounts for the time loss, and anticipates the proper delay to stay on track? How would you collect and present those statistics for monitoring?

• Accessing the GPIO on the kernel side
  • The sysfs interface provides a basic interface to the GPIO ports. How could moving that access to the kernel side with your own kernel module help your product?
  • What functionality would you move from the application into the kernel module (e.g. dit/dah timing)?
  • Would a char block device be better than sysfs?
Extra Credit Exercises - 3

- Finish Peer Mode
  - You can easily use this infrastructure to do peer-to-peer communication without a central server
  - When the Client starts in peer mode, it should start its own server listening, and poll to see if the peer’s server has yet started
  - The client’s main loop would be to send any pending traffic, see if anything came in for its server, and of course check the user input.
  - When the client exits, it then must also stop its server.
  - If the client times out sending to the peer server, then it should close.
  - See if you can find the implementation, complete the missing steps, and start conversing across the room in morse code.
Questions and Answers
Thank you for your participation!
Appendix: Beaglebone Black Setup at Home

Here is how you can build your project for Beaglebone Black

```bash
$ export INSTALL_DIR=`pwd`
$ git clone -b daisy git://git.yoctoproject.org/poky
$ source poky/oe-init-build-env `pwd`/beagle
$ echo 'BBLAYERS += "/path/to/ypdd-adv-layer"' >> conf/bblayers.conf
$ echo 'MACHINE = "beaglebone"' >> conf/local.conf
$ echo 'IMAGE_INSTALL_append = " gdbserver morseapp morsemod openssl"' >> conf/local.conf
$ echo 'EXTRA_IMAGEDEPENDS_append = " gdb-cross"' >> conf/local.conf
$ bitbake core-image-base
```
Appendix: Minnowboard Max Setup at Home

- Here is how you can build your project for Minnowboard Max

```bash
$ export INSTALL_DIR=`pwd`
$ git clone -b daisy git://git.yoctoproject.org/poky
$ git clone -b daisy git://git.yoctoproject.org/meta-intel
$ source poky/oe-init-build-env `pwd`/minnow
$ echo 'BBLAYERS += "$INSTALL_DIR/source/meta-intel"' >> conf/bblayers.conf
$ echo 'BBLAYERS += "/path/to/ypdd-adv-layer"' >> conf/bblayers.conf
$ echo 'MACHINE = "intel-corei7-64"' >> conf/local.conf
$ echo 'IMAGE_INSTALL_append = " gdbserver morseapp morsemod openssl"' >> conf/local.conf
$ echo 'EXTRA_IMAGEDEPENDS_append = " gdb-cross"' >> conf/local.conf
$ bitbake core-image-base
```
Appendix: WandBoard Setup at Home

Here is how you can build your project for WandBoard:

$ mkdir ~/bin
$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo

Create the BSP directory download all of the metadata for the BSP layers

$ PATH=${PATH}:~/bin
$ mkdir fsl-community-bsp
$ cd fsl-community-bsp
$ repo init -u https://github.com/Freescale/fsl-community-bsp-platform -b daisy
$ repo sync
$ ./setup-environment build
$ echo 'BBLAYERS += "/path/to/ypdd-adv-layer"' >> conf/bblayers.conf
$ echo 'MACHINE = "wandboard-quad"' >> conf/local.conf
$ echo 'IMAGE_INSTALL_append = " gdbserver morseapp morsemod openssl"' >> conf/local.conf
$ echo 'EXTRA_IMAGEDEPENDS_append = " gdb-cross"' >> conf/local.conf
$ echo 'ACCEPT_FSL_EULA = "1"' >> conf/local.conf
$
$ bitbake core-image-base
Appendix: Parts List Order Information

• Here is sample ordering information for the breadboard parts. These are all very common and can be obtained from most any source.

<table>
<thead>
<tr>
<th>Part</th>
<th>Sample Ordering Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Breadboard</td>
<td><a href="https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_2155452_-1">https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_2155452_-1</a></td>
</tr>
<tr>
<td>220 ohm resistor</td>
<td><a href="https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_690700_-1">https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_690700_-1</a></td>
</tr>
<tr>
<td>1.2K resistor</td>
<td><a href="https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_690881_-1">https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_690881_-1</a></td>
</tr>
<tr>
<td>2.4K resistor</td>
<td><a href="https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_690953_-1">https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_690953_-1</a></td>
</tr>
<tr>
<td>4.7 K resistor</td>
<td><a href="https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_690924_-1">https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_690924_-1</a></td>
</tr>
<tr>
<td>LED, green</td>
<td><a href="https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_334086_-1">https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_334086_-1</a></td>
</tr>
<tr>
<td>Switch N.O. suitable for breadboard</td>
<td><a href="https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_149948_-1">https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_149948_-1</a></td>
</tr>
<tr>
<td>Jumpers Pin-to-Pin (BeagleBone)</td>
<td><a href="https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_126342_-1">https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_126342_-1</a></td>
</tr>
<tr>
<td>Jumpers Socket-to-Pin (MinnowMax)</td>
<td><a href="https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_2214563_-1">https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_2214563_-1</a></td>
</tr>
</tbody>
</table>
Appendix: Useful Links

- **KObjects / sysfs**
  - http://lwn.net/Articles/266722/

- **Wandboard**

- **Beaglebone SD formatting**
  - http://eewiki.net/display/linuxonarm/BeagleBone+Black#BeagleBoneBlack-SetupmicroSD/SDcard
  - https://www.yoctoproject.org/downloads/bsps/daisy16/beaglebone

- **Parent/child for non-blocking getch**
  - http://www.albany.edu/~csi402/pdfs/handout_15.2.pdf

- **Sockets**
  - http://www.tenouk.com/Module40c.html