LS1021A Enablement

LS1021A-IOT Development

Michael Johnston | Applications Engineer

F E B . 1 0 . 2 0 1 5

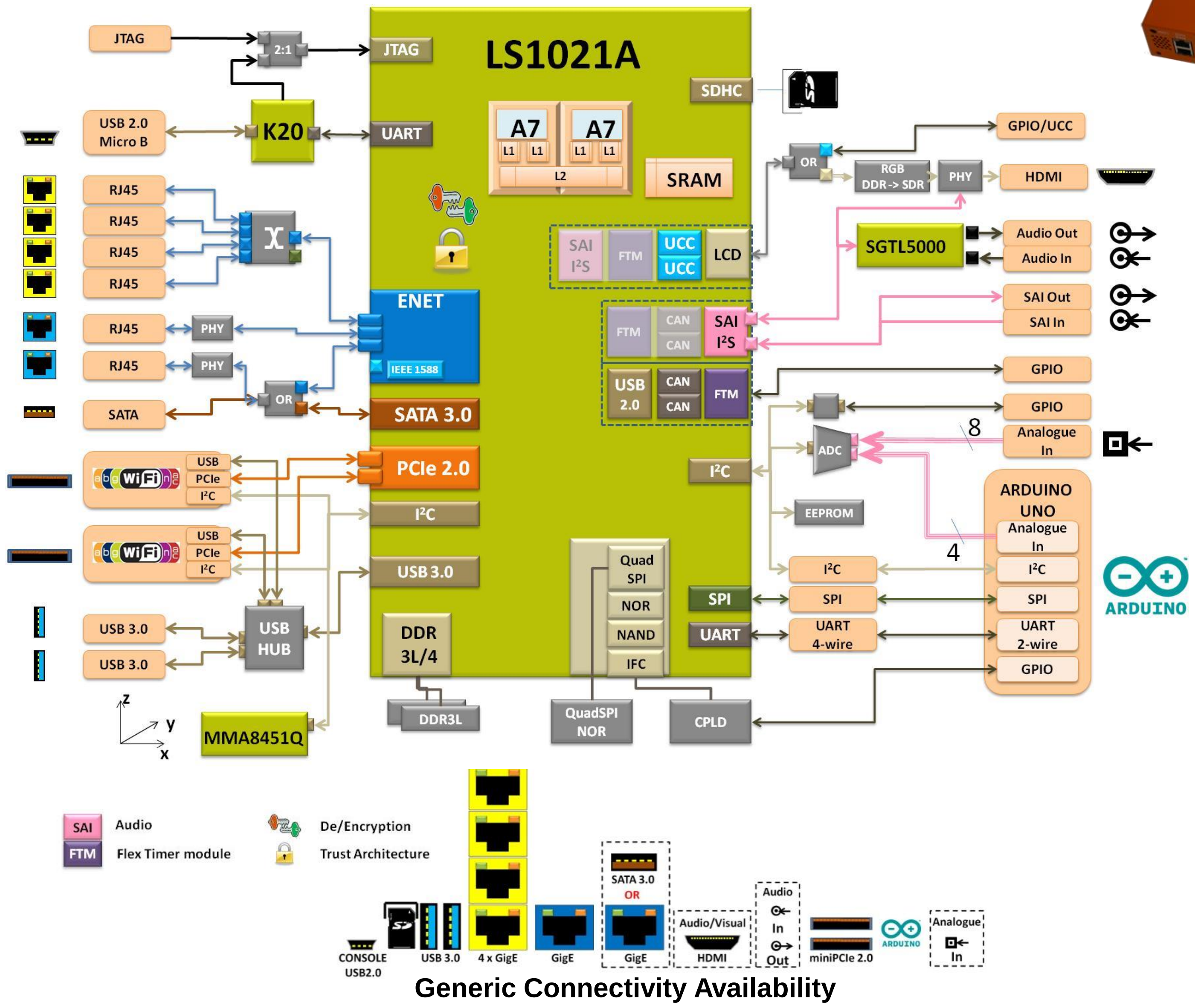
External Use

Agenda

- LS1021A-IOT overview
- Software Support
- Interface Testing
- IOT Gateway Demo
- Proximetry Gateway Demo
- ARM Gateway Demo
- Programmable Logic Controller Demo
- LS1 Community Kit
- Collateral

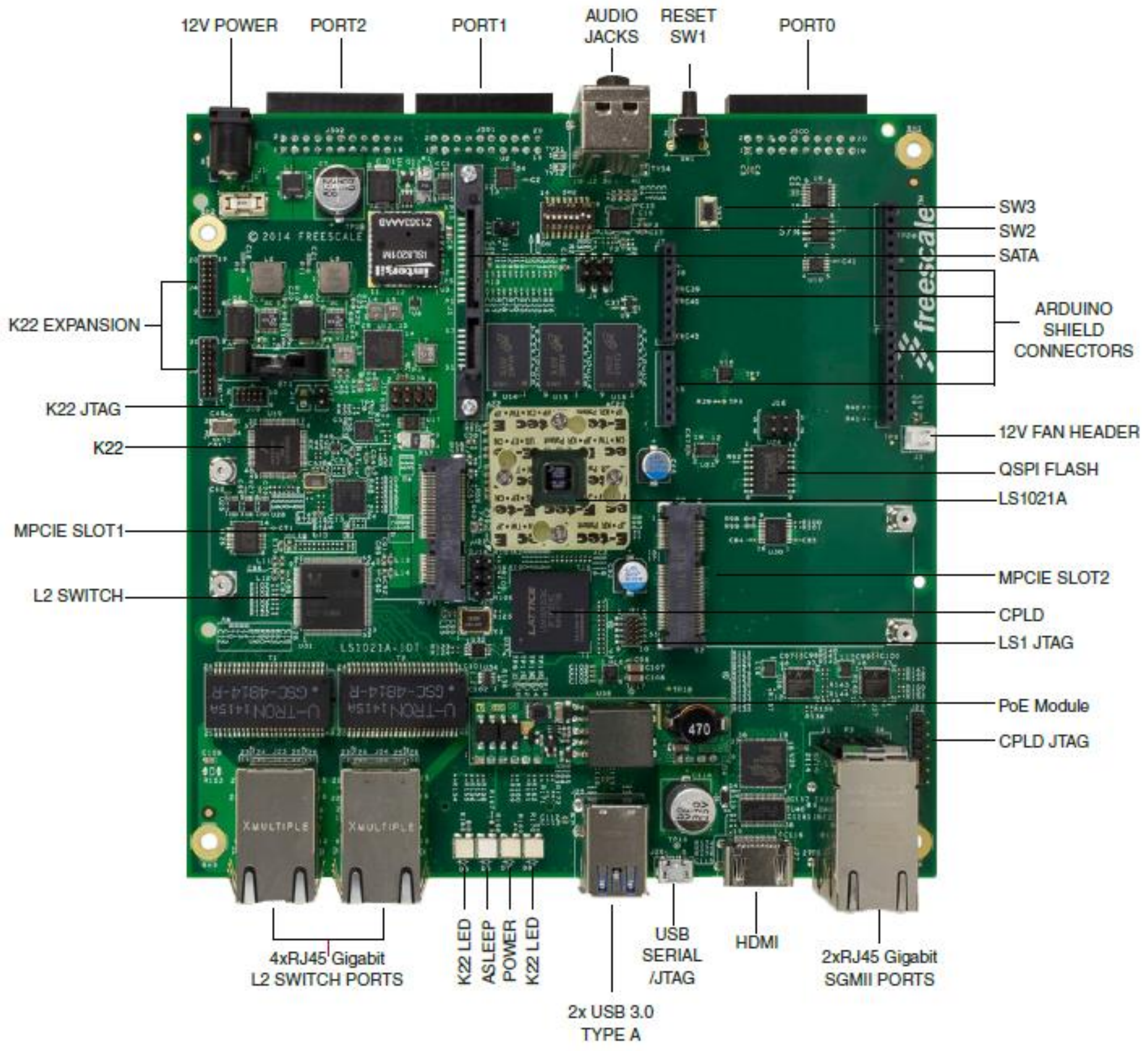
LS1021A-IOT overview

LS1021A-IOT - Block Diagram

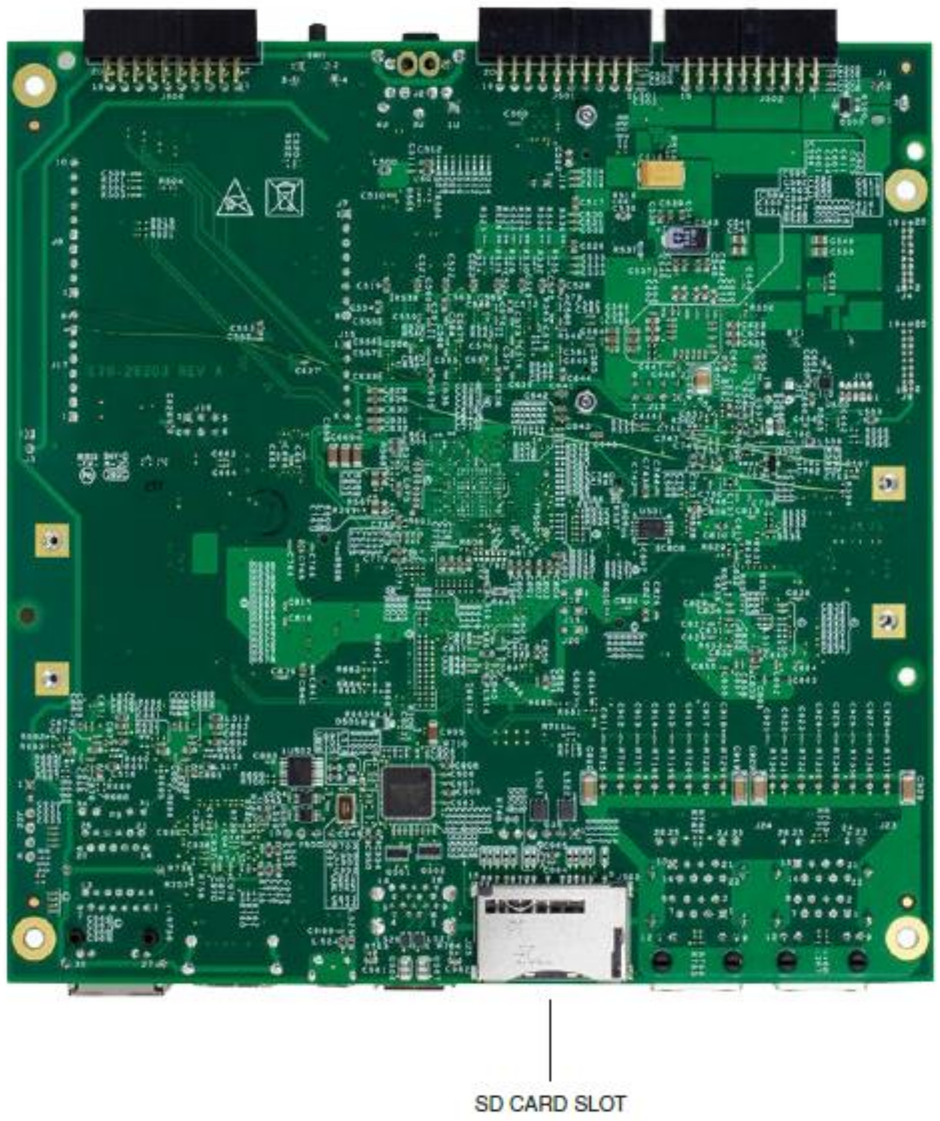


Generic Connectivity Availability

LS1021A-IOT – Top View



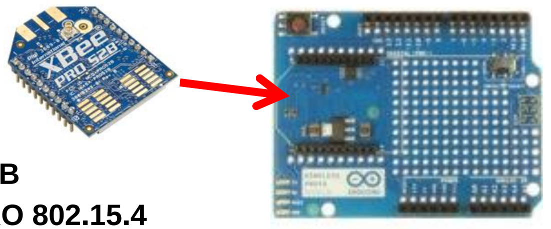
LS1021A-IOT - Flip Side View



Illustrative Arduino/PCIe Modules

Arduino Wireless Proto Shield

XBee® Module



XBee® ZB
XBee-PRO 802.15.4
XBee Wi-Fi
XBee® DigiMesh® 2.4

Arduino GSM Shield

Arduino Proto Shield

Mini PCIe modules

Cellular Modem

Wifi (half size module)

Wifi (Full size module)

LS102x PCI Express 2.0 Controller and SERDES

- 2 PCI Express Controllers can support x1/x2/x4 operation
- USB 3.0 Phy integrates its own dedicated SERDES
- LS1021A-IOT supports options 20 & 70

SRDS_PRTCL_S1[128:135] RCW	A	B	C	D
0	PCIe1(x4)			
80	PCIe2(x2)		PCIe2(x2)	
10	PCIe1 (x1)	SATA1	PCIe2(x2)	
20	PCIe1 (x1)	SGMII1	PCIe2 (x1)	SGMII2
30	PCIe1 (x1)	SATA1	SGMII1	SGMII2
40	PCIe1 (x2)		SATA1	SGMII2
50	PCIe1 (x2)		PCIe2 (x1)	SGMII2
60	PCIe1 (x2)		SGMII1	SGMII2
70	PCIe1 (x1)	SATA1	PCIe2 (x1)	SGMII2

Kinetis K22 USB to UART and JTAG Bridge

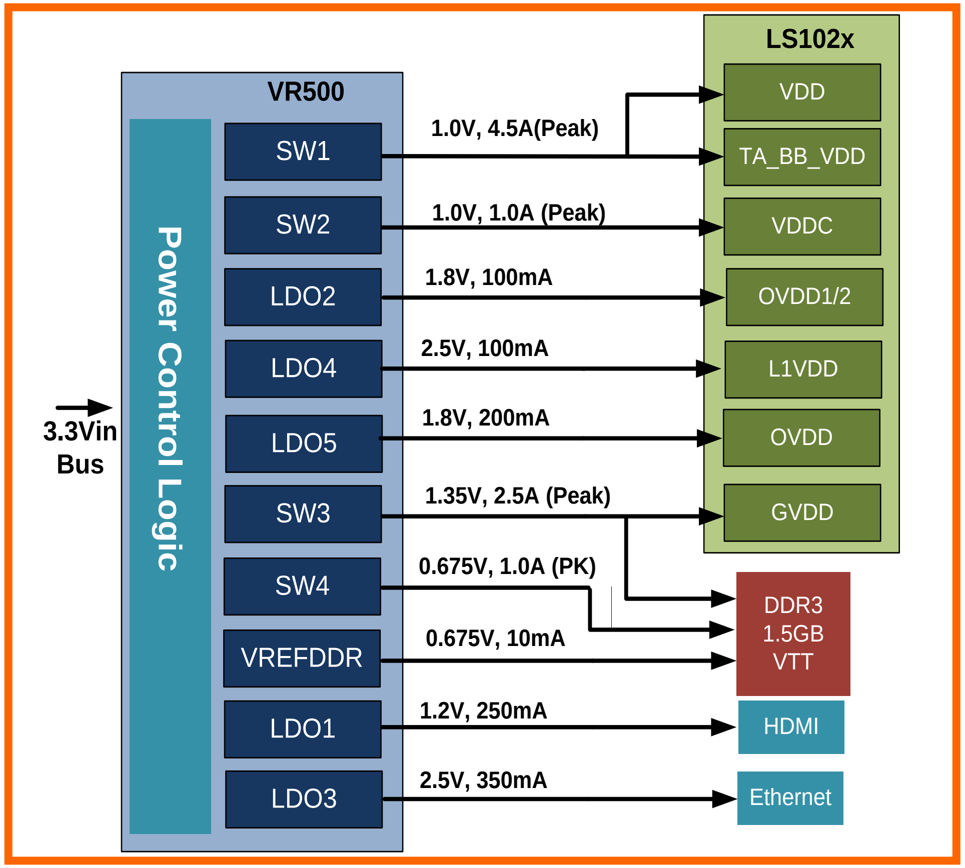
- Features
  - UART to USB conversion
    - Terminal

  - Low Cost on board debug alternative
    - Slower but acceptable
    - Uses ARM CMSIS-DAP on PC
    - Codewarrior Support
    - No external Debug hardware required

  - Additional I/O connectivity to headers
    - Controlled via SPI from LS1021
    - ADC/GPIO etc – remove 3rd party expanders from board
    - Needs Software to be developed.

MC34VR500 PMIC

- Cost effective optimized solution to power the LS1021A
- Features 4 Buck regulators that handle the core and DDR power rails and the five LDOs supply the IO power.
- I2C programmable



Software Support

Software Development Kit (SDK)

- IOT SDK available under Software and Tools Tab at:
http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=LS1021A-IoT

- Packaged and maintained by Software Segment team

- Packages are released ~1 month behind mainline SDK release
  - SDK 1.3 was released 12th December 2014
  - SDK 1.7 due by 16th Feb 2015

- rds build features: OpenWRT, NAT firewall, DLNA, NVR and more

Installing the SDK

- `cat QorIQ-SDK-V1.7-SOURCE-20141218-yocto_RDS_20150206-a* > QorIQ-SDK-V1.7-SOURCE-20141218-yocto_RDS_20150206.iso`
- `sudo mount -o loop QorIQ-SDK-V1.7-SOURCE-20141218-yocto_RDS_20150206.iso /mnt/cdrom/`
- `Run /mnt/cdrom/install`
- `cd ~/QorIQ-SDK-V1.7-20141218-yocto/`
- `./poky/scripts/host-prepare.sh`
- `source ./poky/fsl-setup-poky -m ls1021aiot`
- In /QorIQ-SDK-V1.7-20141218-yocto/build_ls1021aiot_release run:
  - `bitbake fsl-image-rds`

  Other build options such as fsl-image-core are available
- Images will be located in: `build_ls1021aiot_release/tmp/deploy/images/ls1021aiot/`
- U-Boot, kernel and rcw sources can be found in:
  - `build_ls1021aiot_release/tmp/work/ls1021aiot-fsl-linux-gnueabi`

LS1021A-IOT

Boot options

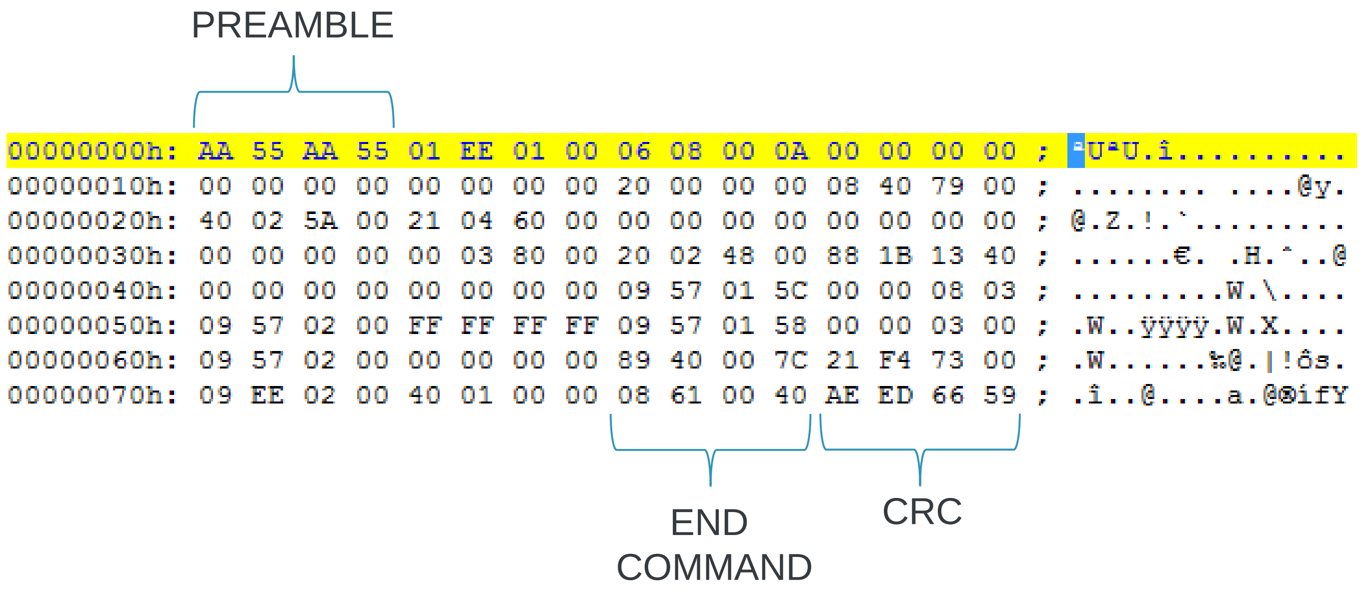
- SD boot requires:
  - RCW, PBI and U-Boot are part of a single binary image (*u-boot-with-spl-pbl.bin*)
  - Linux kernel image (e.g. *uImage-ls1021aiot.bin*)
  - Device tree (e.g. *uImage-ls1021aiot.dtb*)
  - File System (e.g. *fsl-image-core-20150209131232.rootfs.ext2.gz.u-boot)*

- QSPI boot requires:
  - *byte_swap.tcl :* script used to swap the bytes in the binary image into the required format for the QSPI interface
  - RCW/PBI binary (e.g. *rcw_1000_swap.bin)*
  - U-Boot binary image (e.g. *u-boot_swap.bin*)
  - The kernel, dtb and file system could also be swapped and stored in QSPI flash or they could be loaded from another interface

LS1021A-IOT
Flash Boot RCW Image Generation Example

- Create an RCW file, e.g.
  rcw\ls1021aiot\SSR_PPN_20\rcw_1000.rcw

- Run ‘make’ at the top level rcw directory to create the binary, e.g.
  rcw\ls1021aiot\SSR_PPN_20\rcw_1000.bin

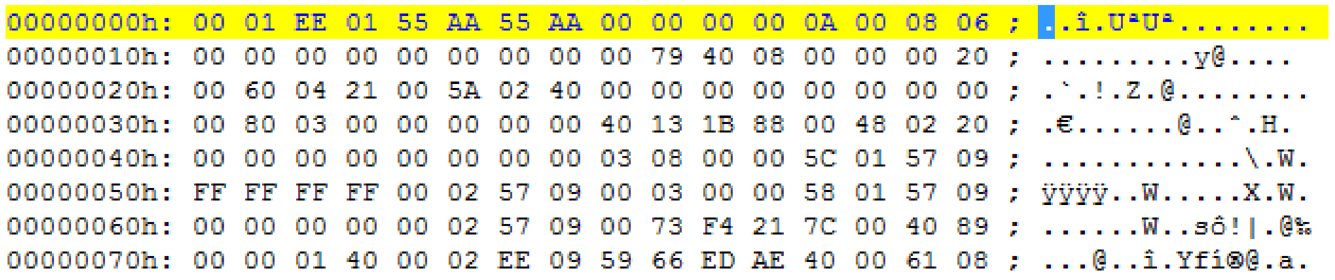


```
00000000h: AA 55 AA 55 01 EE 01 00 06 08 00 0A 00 00 00 00 ; ªUªU.î..........
00000010h: 00 00 00 00 00 00 00 00 20 00 00 00 08 40 79 00 ; ........ ....@y.
00000020h: 40 02 5A 00 21 04 60 00 00 00 00 00 00 00 00 00 ; @.Z.!.`.........
00000030h: 00 00 00 00 00 03 80 00 20 02 48 00 88 1B 13 40 ; ......€. .H.ˆ..@
00000040h: 00 00 00 00 00 00 00 00 09 57 01 5C 00 00 08 03 ; .........W.\....
00000050h: 09 57 02 00 FF FF FF FF 09 57 01 58 00 00 03 00 ; .W..ÿÿÿÿ.W.X....
00000060h: 09 57 02 00 00 00 00 00 89 40 00 7C 21 F4 73 00 ; .W......‰@.|!ôs.
00000070h: 09 EE 02 00 40 01 00 00 08 61 00 40 AE ED 66 59 ; .î..@....a.@®ífY
```

LS1021A-IOT
QSPI Boot Image RCW Programming

- Before programming to the QSPI flash swap the bytes in each double word using the byte_swap script, e.g.

```
tclsh byte_swap.tcl rcw_1000.bin rcw_1000_swap.bin 8
```



```
00000000h: 00 01 EE 01 55 AA 55 AA 00 00 00 00 0A 00 08 06 ; ..î.UªUª........
00000010h: 00 00 00 00 00 00 00 00 00 79 40 08 00 00 00 20 ; .........y@....
00000020h: 00 60 04 21 00 5A 02 40 00 00 00 00 00 00 00 00 ; .`.!.Z.@........
00000030h: 00 80 03 00 00 00 00 00 40 13 1B 88 00 48 02 20 ; .€......@..ˆ.H.
00000040h: 00 00 00 00 00 00 00 00 03 08 00 00 5C 01 57 09 ; ............\.W.
00000050h: FF FF FF FF 00 02 57 09 00 03 00 00 58 01 57 09 ; ÿÿÿÿ..W.....X.W.
00000060h: 00 00 00 00 00 02 57 09 00 73 F4 21 7C 00 40 89 ; ......W..sô!|.@‰
00000070h: 00 00 01 40 00 02 EE 09 59 66 ED AE 40 00 61 08 ; ...@..î.Yfí®@.a.
```

- An easy way to program the QSPI flash is to boot from SD, tftp the binary and then use the U-Boot sf (SPI flash) command, e.g.
  - *tftp rcw_1000-qspiboot_swap.bin*
  - *sf probe*
  - *sf erase 0 0x10000*
  - *sf write 82000000 0 0x100*

LS1021A-IOT

QSPI Boot Image Generation : RCW Naming Conventions

- Default RCW : `SSR_PPN_20`

Legend : **`abc_def_g`**

abc :	eTSEC 1-3 support →	**R** = RGMII, **S** = SGMII	
d :	PCIe 1 support →	**P** =	supported
e :	PCIe 2 support →	**P** =	supported
f :	SATA support →	**S** =	supported
g :	SRDS_PRTCL →	**hex value** of serdes1 protocol	
		N =	not used/not available

- RCW file name convention :

`rcw_x[_clockmode].rcw`

x =	core frequency
clockmode =	if present, means single source clocking for *SYSCLK* and *DDRCLK*

LS1021A-IOT

rcw_1000.rcw example

```
#include
<../ls1021aqds/ls1021a.rcwi>

SYS_PLL_RAT=3
MEM_PLL_RAT=8
CGA_PLL1_RAT=10
SRDS_PRTCL_S1=32
SRDS_PLL_PD_S1=1
SRDS_DIV_PEX=1
USB3_REFCLK_SEL=2
USB3_CLK_FSEL=57
A7_ACE_CLKDIV=2
A7_DBG_CLKDIV=2
HWA_CGA_M1_CLK_SEL=1
PBI_SRC=4
DP_DIV=1
OCN_DIV=1
IFC_MODE=37
DRAM_LAT=1
SYS_PLL_SPD=1
UART_BASE=7
IFC_GRP_A_EXT=1
IFC_GRP_E1_EXT=1
IFC_GRP_F_EXT=1
IFC_GRP_G_EXT=1
```

```
EC1=4
EC2=2
QE-TDMA=6
QE-TDMB=6
SDHC=0
DVDD_VSEL=2
LVDD_VSEL=1
EVDD_VSEL=2
BVDD_VSEL=2

.pbi
// QSPI flash clock
write 0x57015c, 0x00000803
.end
#include
<../ls1021aqds/pcie_link_training.rcw>
#include
<../ls1021aqds/uboot_address.rcw>
```

SCRATCHRW1 (address 1EE_0200)
Used for primary boot image address

LS1021A-IOT
QSPI Boot Image U-Boot Building/Programming

- From U-boot source directory use the command below to build the QSPI image:
  - `make ls1021aiot_qspi_config`
  - `make`

This will generate a u-boot.bin file that can be run against the byte_swap.tcl script.

- Boot the board using SD boot and use the commands below to program U-Boot to the QSPI flash:
  - `tftp u-boot_swap.bin`
  - `sf probe`
  - `sf erase 0x10000 0x80000`
  - `sf write 82000000 0x10000 0x80000`

- At this stage both the RCW and U-Boot will be in QSPI flash
- To boot from QSPI set SW2[1]=ON and reset the board
- At this point users can tftp the standard Linux kernel and file system binaries or run the swap scripts on these binaries and also program them into QSPI flash.
  – The environment variables in U-Boot will have to be modified depending on exactly where the images are loaded from

LS1021A-IOT

SD Boot Image Generation : RCW and U-boot

- Setup the cross compiler, then clean, configure and make the u-boot image file *'u-boot-with-spl-pbl.bin'*
- Copy the image to an SD card, e.g. using the dd command on a Linux host machine

```
$ cd <u-boot source tree>

$ make distclean

$ make ls1021aiot_sdcard_config

$ make

$ sudo dd if=u-boot-with-spl-pbl.bin of=/dev/mmcblk0 bs=512
seek=8 conv=notrunc oflag=dsync
```

- Note:
  - the main board configuration settings can be found in include/configs/ls1021aiot.h
  - the board RCW & PBI settings for SD boot can be found in board/freescale/ls1021aiot/ls1_rcw.cfg & board/freescale/ls1021aiot/ls102xa_pbi.cfg

LS1021A-IOT – U-Boot output

```
U-Boot 2014.01Layerscape-SDK-V1.3+g50d6848 (Dec 01 2014 - 13:42:50)

CPU:   Freescale LayerScape LS1021E, Version: 1.0, (0x87081110)
Clock Configuration:
       CPU0(ARMV7):1000 MHz,
       Bus:300  MHz, DDR:800  MHz (1600 MT/s data rate),
Reset Configuration Word (RCW):
       00000000: 0608000a 00000000 00000000 00000000
       00000010: 20000000 08407900 60025a00 21046000
       00000020: 00000000 00000000 00000000 00038000
       00000030: 20024800 881b1340 00000000 00000000
Board: LS1021AIOT
CPLD:  Unknown (23000000)
I2C:   ready
DRAM:  1 GiB (DDR3, 32-bit, CL=10.5, ECC off)
MMC:   FSL_SDHC: 0
In:    serial
Out:   serial
Err:   serial
SATA link 0 timeout.
AHCI 0001.0300 1 slots 1 ports ? Gbps 0x1 impl SATA mode
flags: 64bit ncq pm clo only pmp fbss pio slum part ccc
scanning bus for devices...
Found 0 device(s).
Net:   Phy not found
PHY reset timed out
eTSEC1, eTSEC2 [PRIME], eTSEC3
Hit any key to stop autoboot:  0
=>
```

Interface Testing

Ethernet

- iperf network testing tool used for soak testing the board
- Included in Yocto file system, e.g. part of fsl-image-core build
- SET-UP
  - On host PC side:
    - iperf -s
  - On LS1021A-IOT:
    - iperf -c *hostip* -i 5 -t 10000, e.g.

```
root@ls1021aqds:~# iperf -c 192.168.1.1 -i 5 -t 10000
------------------------------------------------------------
Client connecting to 192.168.1.1, TCP port 5001
TCP window size: 20.7 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.9 port 54807 connected with 192.168.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 5.0 sec    483 MBytes    810 Mbits/sec
```

L2 switch

- RGMII link from LS1021A to Realtek RTL8365MB

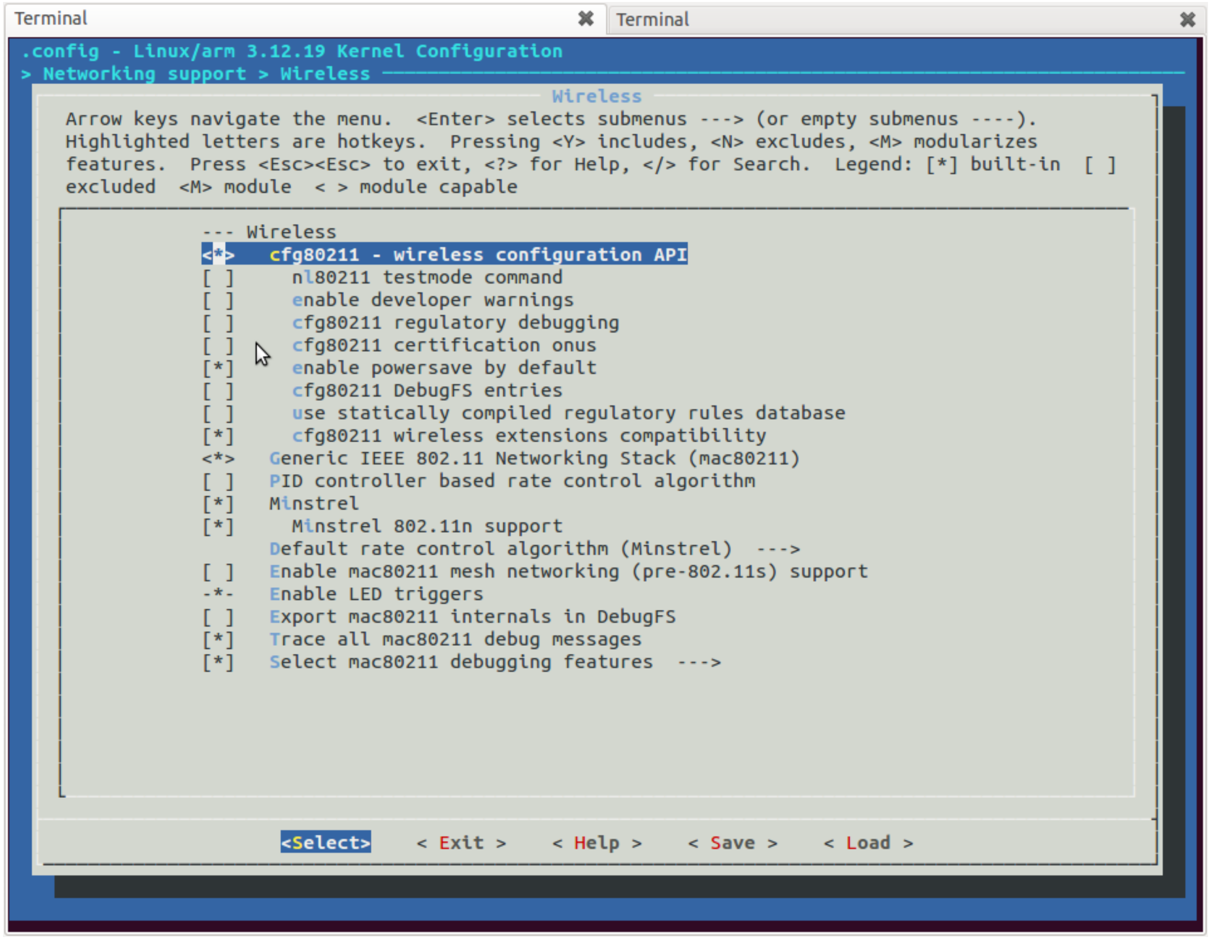
- SPI interface used for switch configuration
  - Created user space app that calls the Realtek switch API and LS1 SPI reads/writes to set-up the fixed 1Gbps RGMII link
    - rtlinit app included in IOT SDK (in SDK 1.3 only rds build has this included)

  - Without this configuration the switch will forward data across the 4 ports but no traffic can be sent to/from LS1021A on these ports

- Linux ifconfig shows the interface as eth2

Linux WLAN support

- By default the IOT kernel image will contain support for the 802.11 stack and APIs:



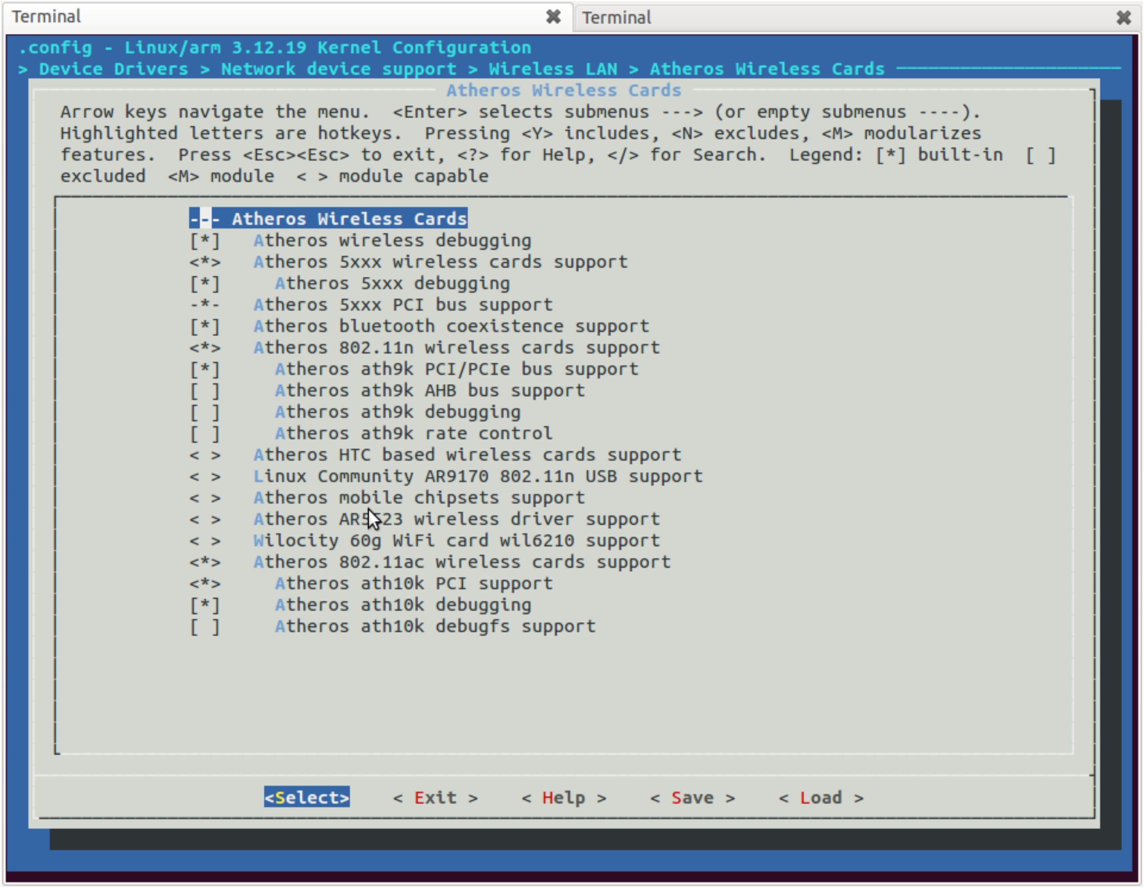
```
Terminal                                          Terminal
.config - Linux/arm 3.12.19 Kernel Configuration
> Networking support > Wireless ─────────────────────────────────────
                           Wireless
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]
  excluded  <M> module  < > module capable

            --- Wireless
            <*>   cfg80211 - wireless configuration API
            [ ]     nl80211 testmode command
            [ ]     enable developer warnings
            [ ]     cfg80211 regulatory debugging
            [ ]     cfg80211 certification onus
            [*]     enable powersave by default
            [ ]     cfg80211 DebugFS entries
            [ ]     use statically compiled regulatory rules database
            [*]     cfg80211 wireless extensions compatibility
            <*>   Generic IEEE 802.11 Networking Stack (mac80211)
            [ ]   PID controller based rate control algorithm
            [*]   Minstrel
            [*]     Minstrel 802.11n support
                  Default rate control algorithm (Minstrel)  --->
            [ ]   Enable mac80211 mesh networking (pre-802.11s) support
            -*-   Enable LED triggers
            [ ]   Export mac80211 internals in DebugFS
            [*]   Trace all mac80211 debug messages
            [*]   Select mac80211 debugging features  --->

          <Select>    < Exit >    < Help >    < Save >    < Load >
```

Linux WLAN support – compatible WLAN cards

- By default the IOT images will contain support for various WLAN cards, e.g. Atheros 9k
  - Other compatible devices can be added under the Linux ‘make menuconfig’ Wireless LAN device driver menu:



```
.config - Linux/arm 3.12.19 Kernel Configuration
> Device Drivers > Network device support > Wireless LAN > Atheros Wireless Cards
                             Atheros Wireless Cards
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]
  excluded  <M> module  < > module capable

            --- Atheros Wireless Cards
            [*]   Atheros wireless debugging
            <*>   Atheros 5xxx wireless cards support
            [*]     Atheros 5xxx debugging
            -*-   Atheros 5xxx PCI bus support
            [*]   Atheros bluetooth coexistence support
            <*>   Atheros 802.11n wireless cards support
            [*]     Atheros ath9k PCI/PCIe bus support
            [ ]     Atheros ath9k AHB bus support
            [ ]     Atheros ath9k debugging
            [ ]     Atheros ath9k rate control
            < >   Atheros HTC based wireless cards support
            < >   Linux Community AR9170 802.11n USB support
            < >   Atheros mobile chipsets support
            < >   Atheros AR5523 wireless driver support
            < >   Wilocity 60g WiFi card wil6210 support
            <*>   Atheros 802.11ac wireless cards support
            <*>     Atheros ath10k PCI support
            [*]     Atheros ath10k debugging
            [ ]     Atheros ath10k debugfs support

          <Select>    < Exit >    < Help >    < Save >    < Load >
```

Linux WLAN support – firmware support

- Many of the WLAN cards require a firmware download

- This is added to the IOT file system by default using the linux-firmware option within the bitbake build.

- Verify correct operation during Linux boot, e.g.

```
[   25.296489] iwlwifi 0001:01:00.0: loaded firmware version 22.0.7.0 op_mode iwlmvm
[   25.303995] iwlwifi 0001:01:00.0: Detected Intel(R) Dual Band Wireless AC 7260, REV=0x144
```

- Confirm ifconfig -a reports wlan0 as available:

```
wlan0     Link encap:Ethernet  HWaddr c0:4a:00:f4:34:47
          inet addr:192.168.0.3  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::c24a:ff:fef4:3447/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:808 (808.0 B)
```

Linux WLAN support – setting WLAN parameters

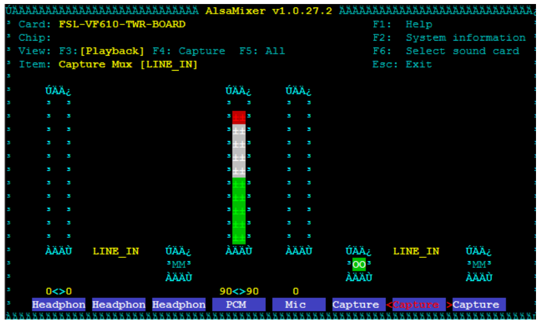
- hostapd
  - user space daemon for wireless access point and authentication servers
    - hostap-daemon can be included as part of Yocto file system build
- In the /etc/hostapd.conf file users can set parameters such as:
  - interface=wlan0
  - ssid=LS1021A-IOT-DEMO
  - hw_mode=g
  - channel=6
  - wpa=2
  - wpa_passphrase=fsl_iot_demo
- To test the connection 2 PCs can be connected to the AP and iperf used to transfer traffic over wlan0

Cellular

- Telit HE910 3G mPCI module used in IOT testing (uses USB signals on mPCI connector)
- In kernel options support is enabled using Device Drivers->USB Support->USB modem (CDC ACM) support.
  - CONFIG_USB_SERIAL=y
  - CONFIG_USB_SERIAL_GENERIC=y
  - CONFIG_PPP=y
  - CONFIG_PPP_ASYNC=y
- Tested using pppd_script and hsdpa_connect configuration scripts
  - Then run `pppd file /etc/pppd_script &`
- Once the connection is established ifconfig will show:

```
ppp0      Link encap:Point-to-Point Protocol
inet addr:10.145.0.193  P-t-P:10.145.0.193  Mask:255.255.255.255
UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
RX packets:8 errors:0 dropped:0 overruns:0 frame:0
TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:3
RX bytes:122 (122.0 B)  TX bytes:164 (164.0 B)
```

Audio

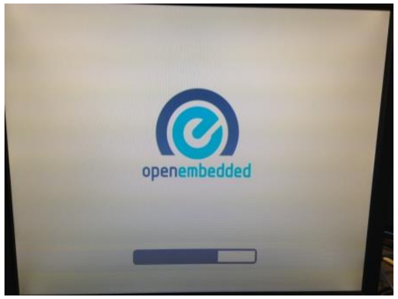


```
AlsaMixer v1.0.27.2
Card: FSL-VF610-TWR-BOARD                          F1:  Help
Chip:                                              F2:  System information
View: F3:[Playback] F4: Capture  F5: All           F6:  Select sound card
Item: Capture Mux [LINE_IN]                        Esc: Exit

         LINE_IN                                    LINE_IN
  0<>0                   90<>90     0
Headphon Headphon Headphon  PCM     Mic   Capture <Capture >Capture
```

- LS1021A-IOT supports line in/out
- Tested with Advanced Linux Sound Architecture (ALSA) support in the kernel
  - Device drivers->Sound card support->Advanced Linux Sound Architecture->ALSA for SoC Audio Support->SoC Audio for Freescale VF610 CPUs
  - This will enable Freescale Serial Audio Interface (SAI) and codec support
- Use arecord/aplay to test record and playback features, e.g.
  - `arecord -f S16_LE -r 44100 -t wav -c2 testfile`
  - `aplay -f S16_LE -r 44100 -t wav -c2 testfile`
- Configuration can be modified by calling alsamixer or automatically at start-up by updating /var/lib/alsa/asound.state

HDMI



- LS1021A-IOT supports HDMI to display using Silicon image HDMI transmitter

- Kernel config includes support for Silicon image HDMI transmitter SiI9022:
  - Device drivers->Graphics support->Support for frame buffer devices->Si Image SII9022 DVI/HDMI Interface Chip

- U-Boot othbootargs set to hdmi

- Tested with OpenEmbedded file system included in Yocto build
  - bitbake fsl-image-x11

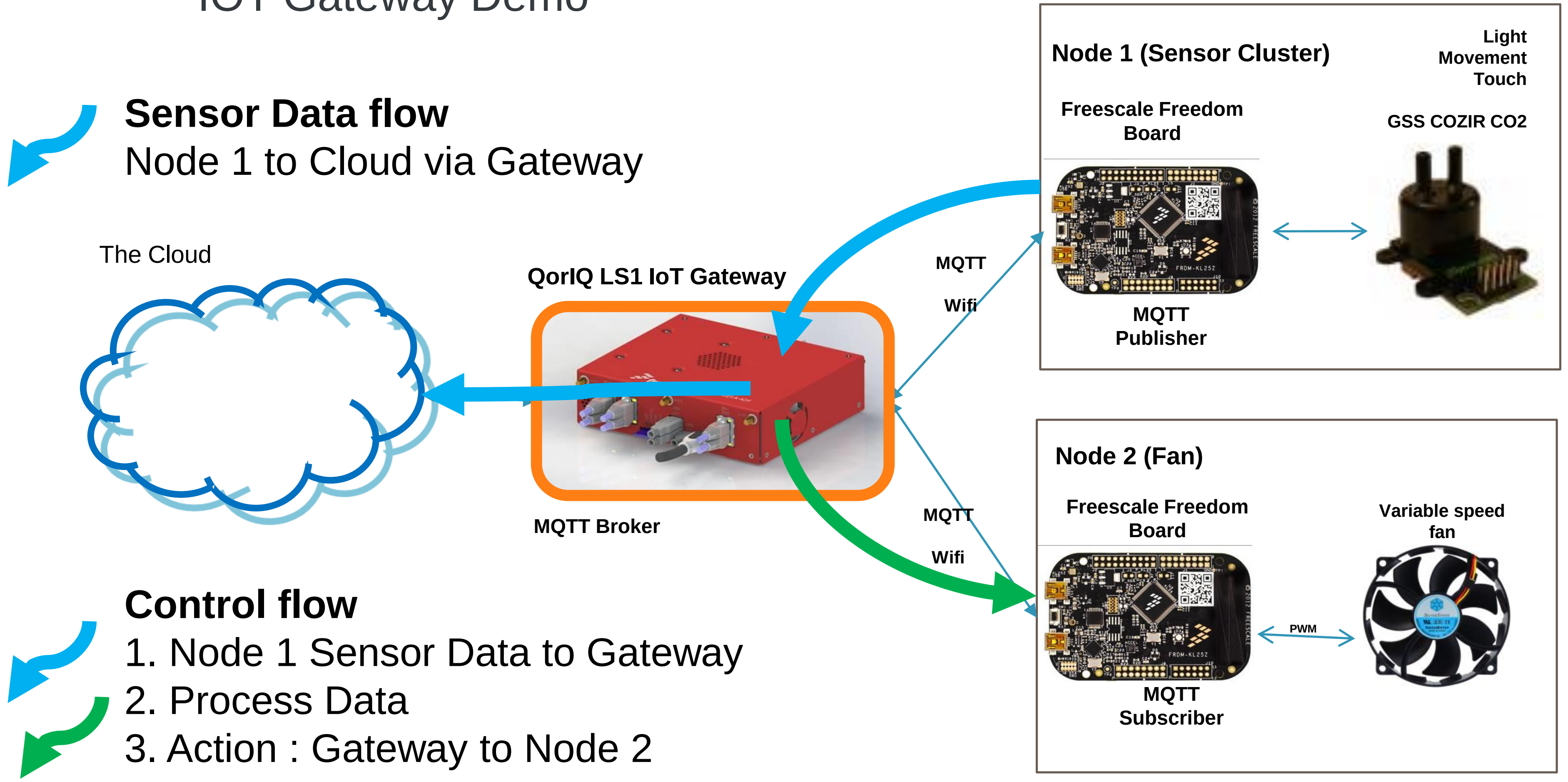
IOT Gateway Demo

IOT Gateway demo – Node.js, Node-RED & MQTT

- Node.js
  - Open source runtime environment for JavaScript applications
  - Part of Yocto build so very easy to add
- Node-RED
  - Open source software that runs on Node.js. Provides a browser based graphical programming tool for creating applications. Programs are built upon 'nodes' – blocks that provide specific functionality e.g. subscribing to a MQTT topic
  - Sources available from http://nodered.org/
- MQTT (Message Queuing Telemetry Transport)
  - Mosquitto is an open source implementation of the MQTT protocol.
  - MQTT is a simple, lightweight messaging protocol that runs on top of TCP/IP. It consists of publishers (components of the network that create data), subscribers (components of the network that consume data) and a broker.
  - Publishers publish data to topics hosted on a broker which then redistributes the data to any clients subscribed to the topic.
  - Source available from http://mosquitto.org/download/
- Currently creating recipes for Mosquitto and Node-RED to simplify the demo build process
  - App note planned to document the process

IOT Gateway Demo

Sensor Data flow
Node 1 to Cloud via Gateway



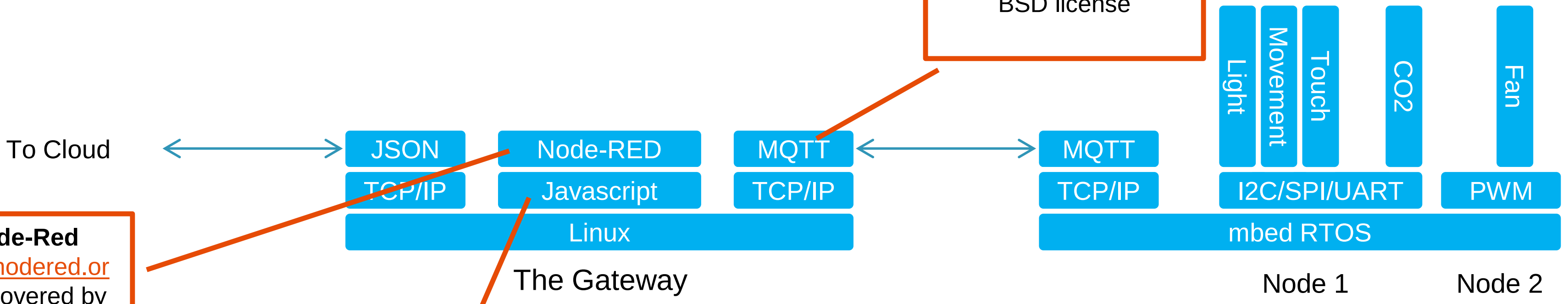
Control flow
1. Node 1 Sensor Data to Gateway
2. Process Data
3. Action : Gateway to Node 2

The Demo – License Considerations
The Demo – Communication

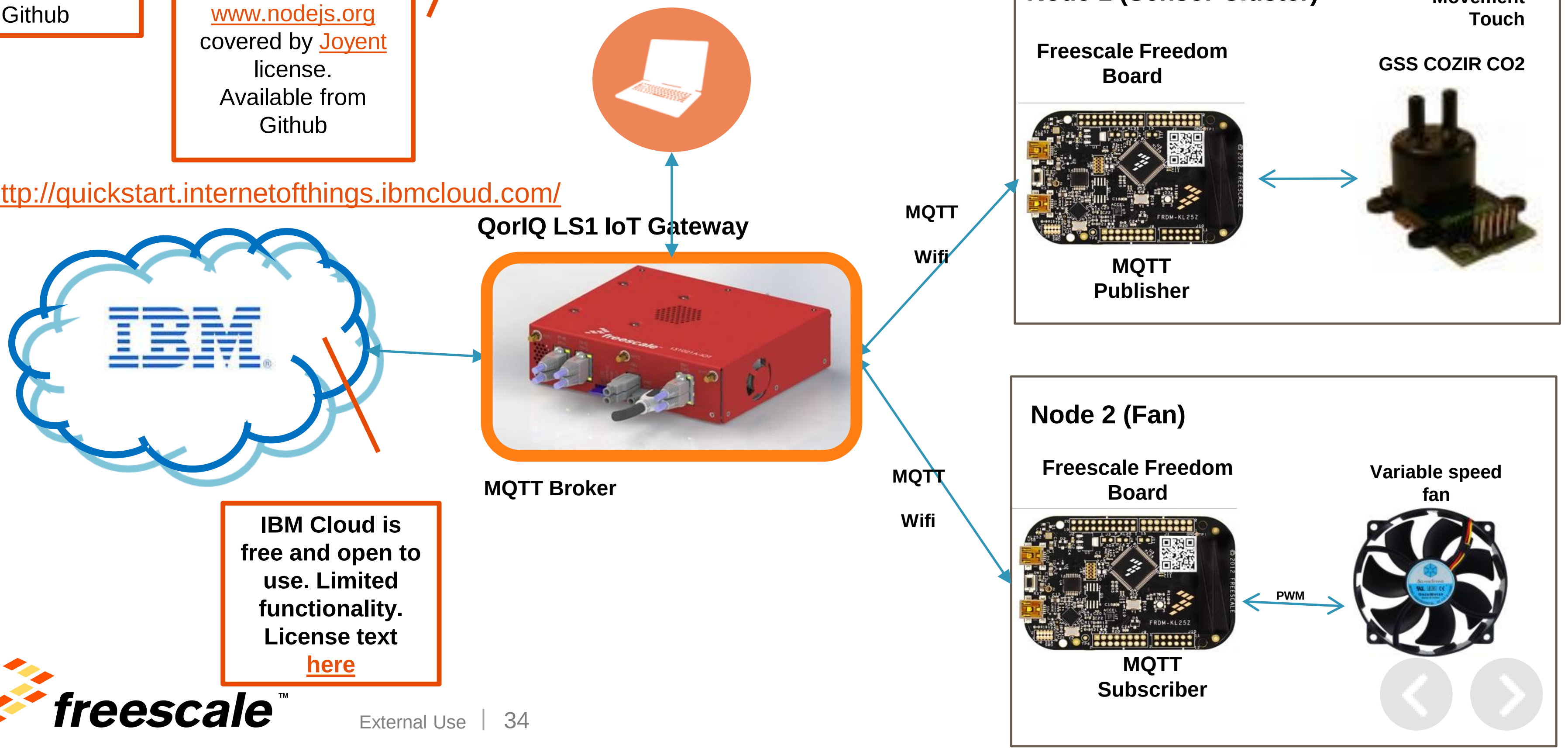
MQTT from http://mosquitto.org/ is BSD license

Node-Red www.nodered.org is covered by Apache2.0 license and available via Github

node.js www.nodejs.org covered by Joyent license. Available from Github

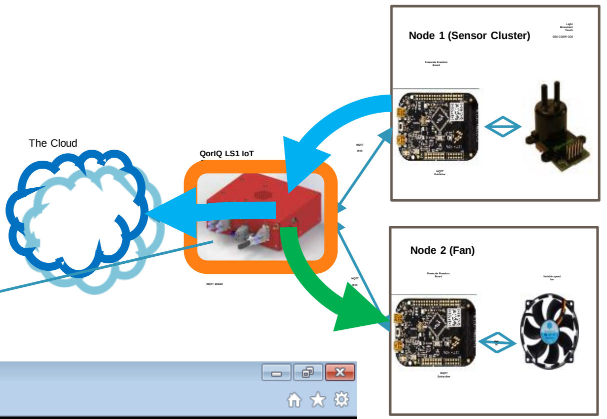


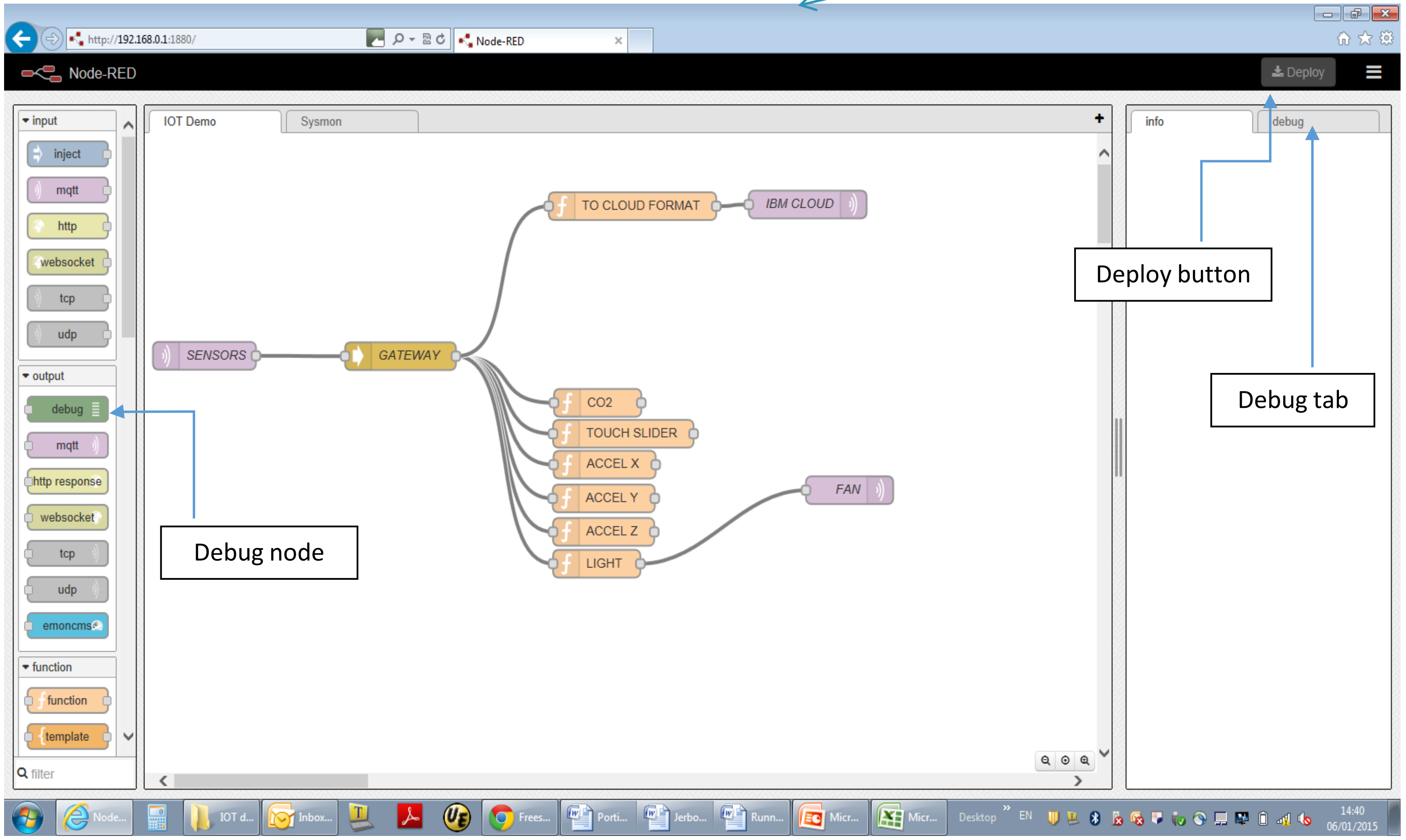
http://quickstart.internetofthings.ibmcloud.com/



IBM Cloud is free and open to use. Limited functionality. License text here

Node-RED Development Environment on LS1 IoT Gateway





Kinetis nodes

- The MQTT publisher and subscriber nodes are implemented with using a FRDM-KL25Z Kinetis processor board paired with an Avnet Wi-Go board, which provides additional Wi-Fi and sensor functionality.

- Both publisher and subscriber nodes are implemented on ARM's mbed platform. This environment provides drivers for the Wi-Fi chipset and libraries to read values from each of the sensors on the platform. MQTT client functionality comes from the Paho MQTT library that is integrated with mbed.

- All data is sent in JSON format to the gateway. An example of this format is shown below:

```
{"temperature_sensor": 65, "light_sensor":120, "touch sensor": 50}
```

- Example code for the Avnet Wi-Go (code covering both sensors and Wi-Fi) board can be found here: http://developer.mbed.org/cookbook/Wi-Go

- Example program for the MQTT client can be found here:
- http://developer.mbed.org/teams/IBM_IoT/code/IBMIoTClientEthernetExample/

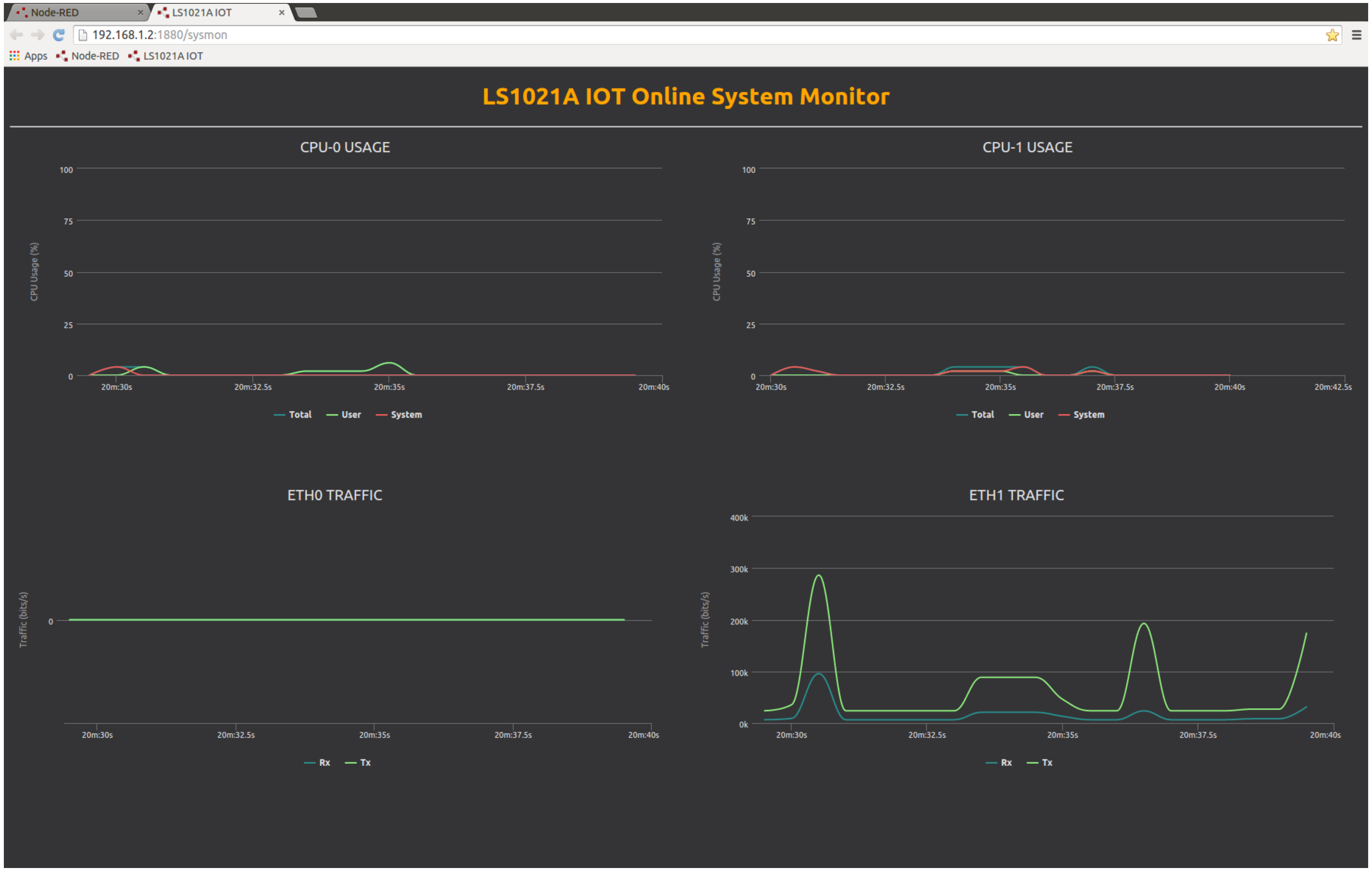
- Example projects for sensor and fan nodes available on request
  - Create a user account at http://developer.mbed.org/ to build/modify

Demo Kit (Wifi based)

- KL25 MQTT Publisher ✓
- KL25 MQTT Subscriber ✓
- QorIQ P1 MQTT Broker ✓
- Fan integration ✓
- CO2 Sensor integration ✓
- IBM Cloud integration ✓

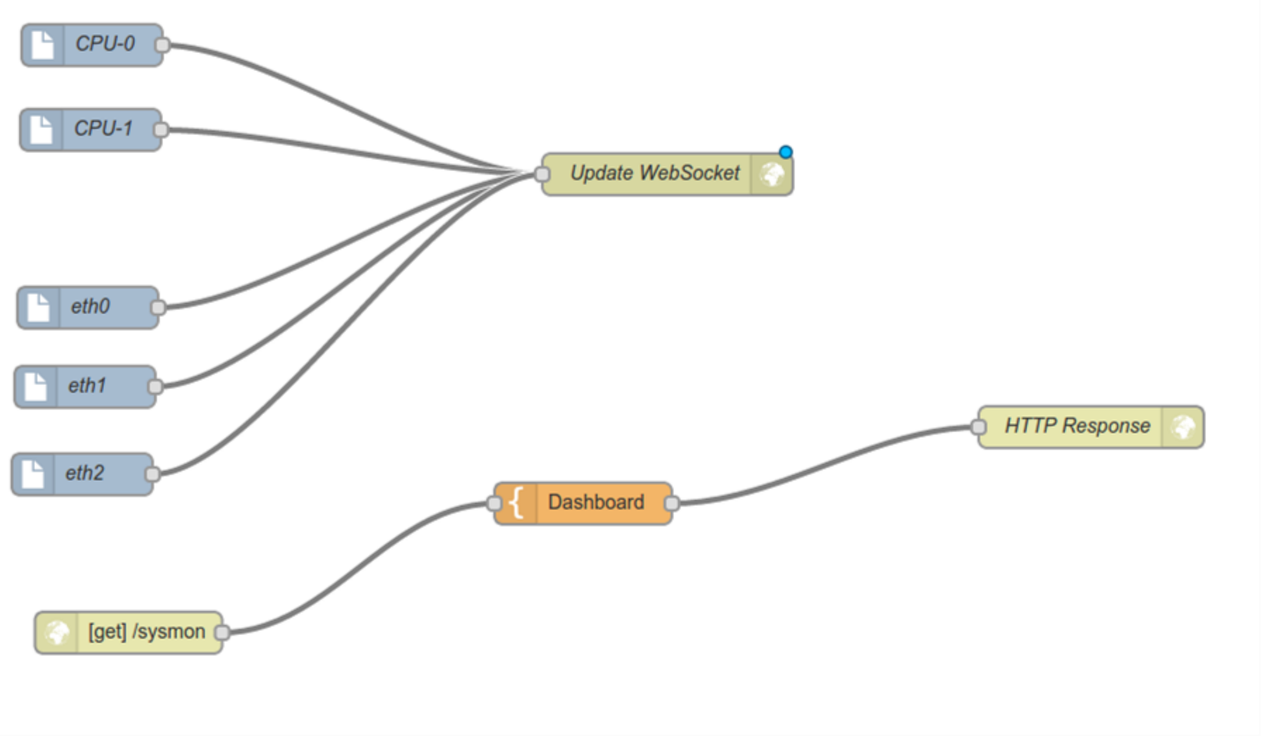
- QorIQ LS1 MQTT Broker ✓
- QorIQ LS1 Node-RED ✓

System monitor option using Node-RED

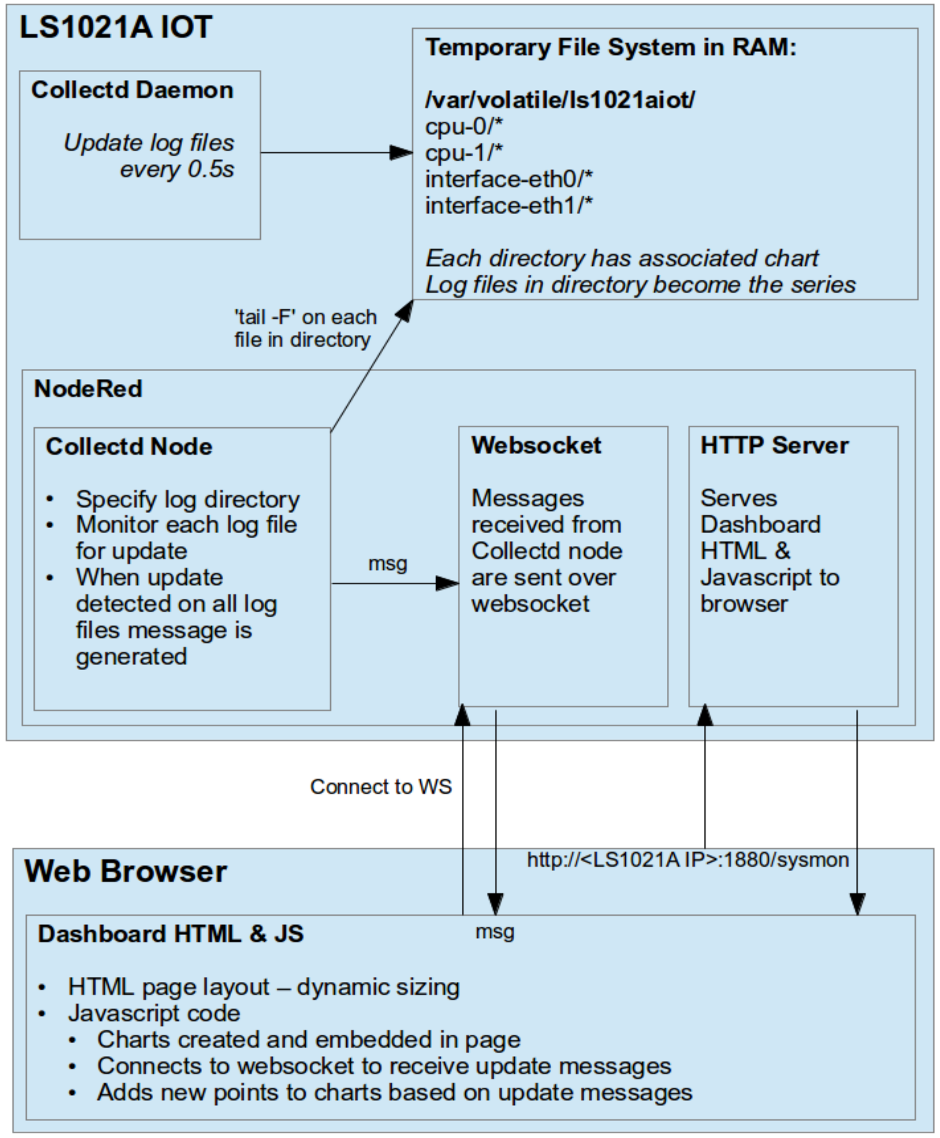


Using Node-RED – sysmon example

- Three components interact to enable the web front-end to chart real time performance data:
  - The Collectd daemon running on LS1021A
  - The Collectd node within the NodeRed server environment on LS1021A
  - The chart dashboard web page running in a web browser



Sysmon flow



LS1021A IOT
Collectd Daemon
Update log files every 0.5s
Temporary File System in RAM:
/var/volatile/ls1021aiot/
cpu-0/*
cpu-1/*
interface-eth0/*
interface-eth1/*
Each directory has associated chart
Log files in directory become the series
'tail -F' on each file in directory
NodeRed
Collectd Node
Specify log directory
Monitor each log file for update
When update detected on all log files message is generated
msg
Websocket
Messages received from Collectd node are sent over websocket
HTTP Server
Serves Dashboard HTML & Javascript to browser
Connect to WS
http://<LS1021A IP>:1880/sysmon
Web Browser
msg
Dashboard HTML & JS
HTML page layout – dynamic sizing
Javascript code
Charts created and embedded in page
Connects to websocket to receive update messages
Adds new points to charts based on update messages

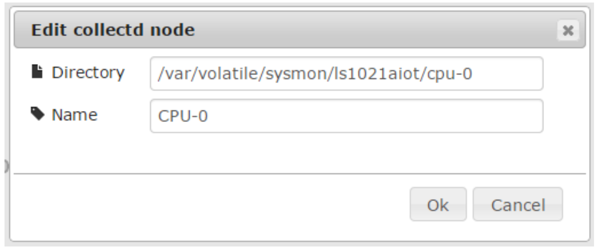
Collectd

- Open source Linux daemon that periodically collects system performance statistics
- Cross compiled for LS1. Extract demo package to IOT file system.
- Functionality comes in the form of plugins that can be included and excluded from the build
- IOT configuration loads only plugins that require no external libraries, relying only on the */proc* filesystem component of the Kernel
  - /proc gives access to runtime system information (e.g. system memory, devices mounted, hardware configuration, etc).
- Available plugins can be loaded and configured with the collectd.conf script that is used when Collectd is started (/etc/collectd.conf)
  - Hostname "ls1021aiot“
  - Interval 0.5
  - LoadPlugin cpu
  - LoadPlugin csv
  - LoadPlugin interface

```
<Plugin csv>
            DataDir "/var/volatile/sysmon"
            StoreRates true
</Plugin>
```

```
<Plugin interface>
            Interface "lo"
            Interface "sit0"
            Interface "can0"
            Interface "can1"
            IgnoreSelected true
</Plugin>
```

The NodeRed 'Collectd' Node

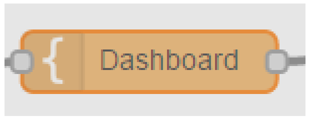

Edit collectd node

Directory: /var/volatile/sysmon/ls1021aiot/cpu-0

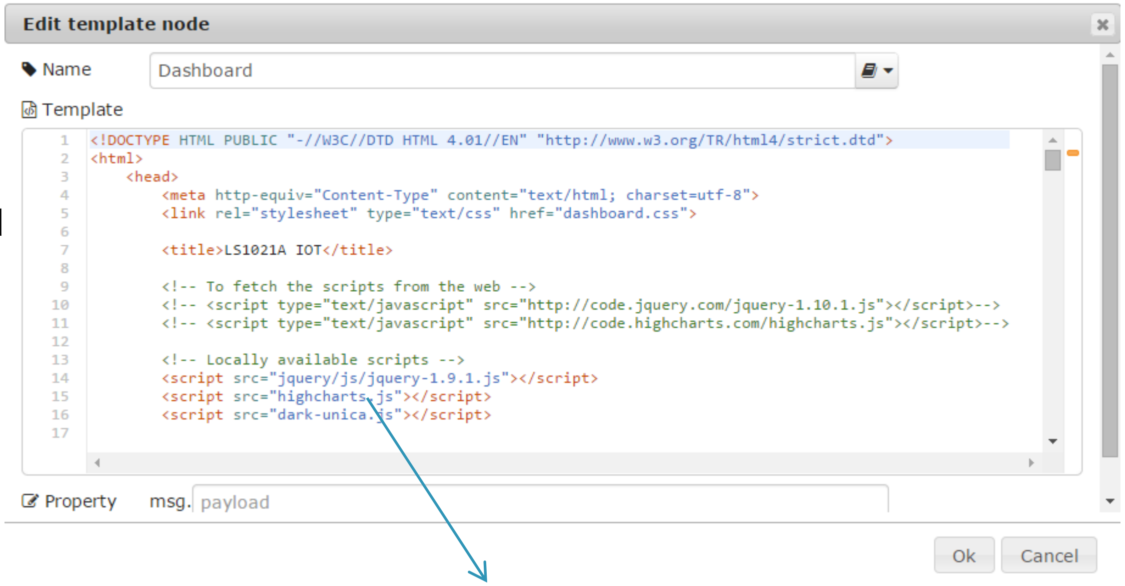
Name: CPU-0

- A custom NodeRed input node (written in node.js Javascript) was created to monitor Collectd log files and generate a message whenever a new entry is added to the log.
- Each instance of the Collectd node operates on a directory containing one or more log files.
- The Collectd node is not intrinsically linked to Collectd. As long as the directory specified contains one or more text files that are periodically added to, the Collectd node can monitor these files and generate Node-Red messages each time a line is added.
- When it has registered a change in ALL of the files being monitored in a directory, a message is generated containing the new lines added. This message can be used downstream to generate a chart.
- In the NodeRed flow, each Collectd node has a connection to the *Update Socket* websocket-out node. The websocket-out node creates a websocket that can be connected to by web pages or services. The chart dashboard web page subscribes to this websocket. Each time the *Update Socket* node receives a message from a Collectd node, it pushes the message over the web socket to all subscribers.

The Web Dashboard



- NodeRed contains simple HTTP server functionality that can be used to respond to HTTP requests.
  - used to serve the HTML/Javascript that constitutes the chart dashboard to a browser connecting to <LS1021A IPAddr>:1880/sysmon.
  - NodeRed simply responds to GET requests to this URL with the entire HTML/JS of the web dashboard
- Changes can be made directly in the editor and immediately deployed
- The Javascript:
  - intialises the charts on the page
  - connects to the websocket
  - handles received messages and updates the appropriate chart (received message has the 'topic' property that indicates the chart it applies to)
  - the payload is parsed to extract the data and add it to the appropriate series on the chart.



Edit template node

Name: Dashboard

Template

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
        <link rel="stylesheet" type="text/css" href="dashboard.css">

        <title>LS1021A IOT</title>

        <!-- To fetch the scripts from the web -->
        <!-- <script type="text/javascript" src="http://code.jquery.com/jquery-1.10.1.js"></script>-->
        <!-- <script type="text/javascript" src="http://code.highcharts.com/highcharts.js"></script>-->

        <!-- Locally available scripts -->
        <script src="jquery/js/jquery-1.9.1.js"></script>
        <script src="highcharts.js"></script>
        <script src="dark-unica.js"></script>

```

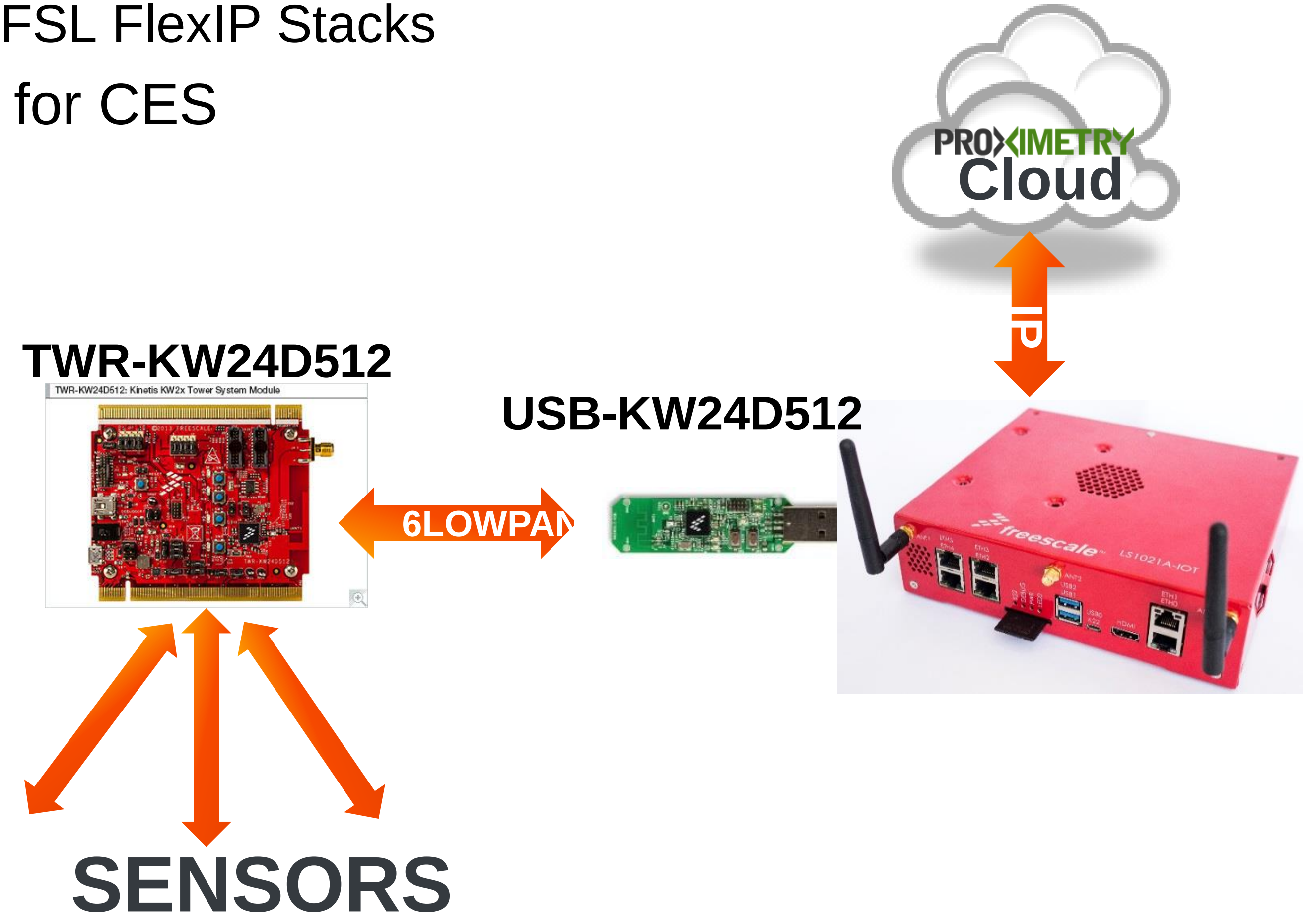
Property: msg. payload

used for the chart graphics

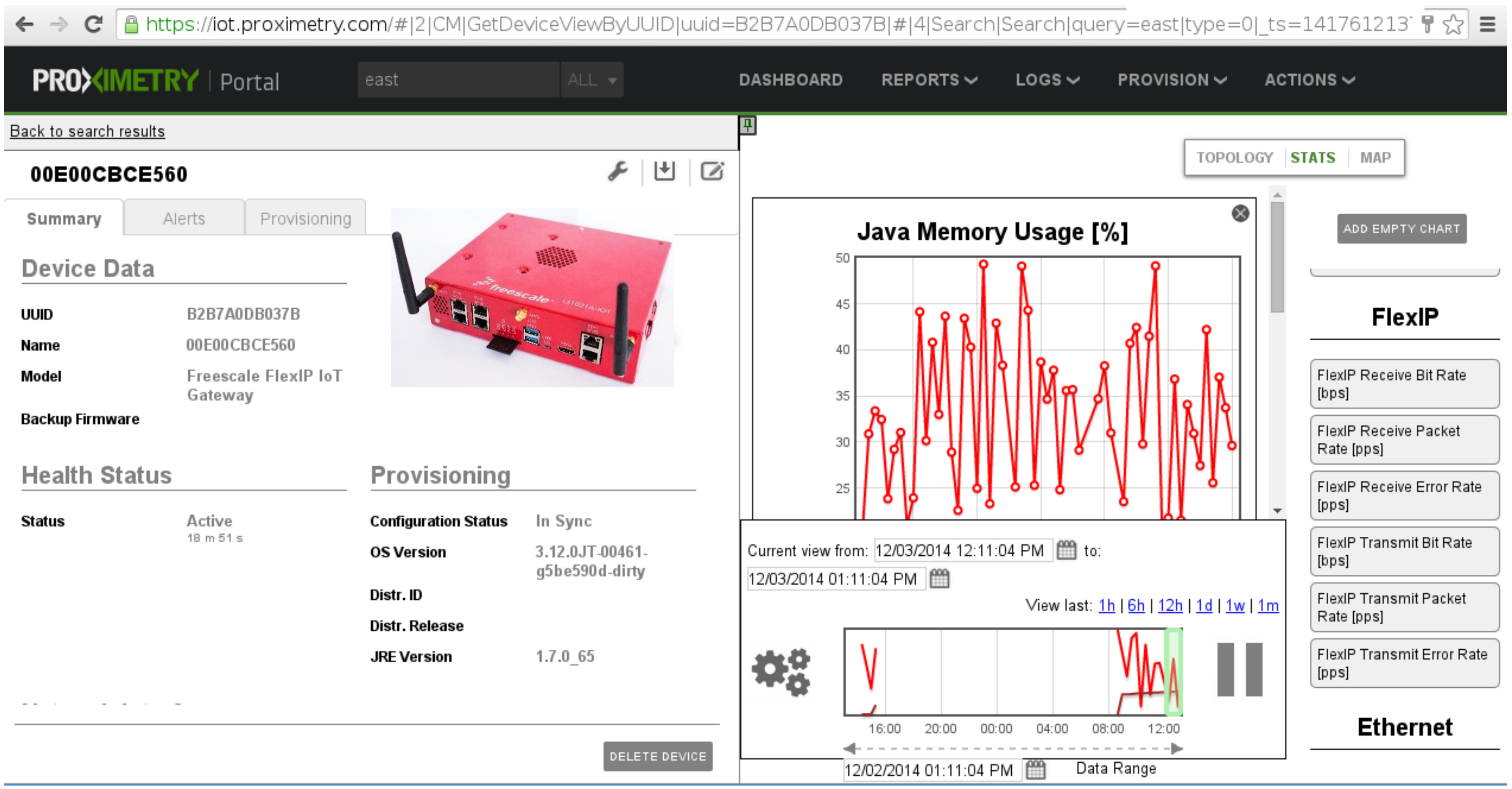
Proximetry IOT Demo

Freescale 6LowPAN based IOT System Demo

- Proximetry produce software solutions for management of IOT networks
- Leverage existing FSL iMX onebox Proximetry demo
  - Uses FSL FlexIP Stacks
- Ready for CES



Proximetry Cloud Interface



00E00CBCE560

Device Data

UUID	B2B7A0DB037B
Name	00E00CBCE560
Model	Freescale FlexIP IoT Gateway
Backup Firmware	

Health Status

Status	Active 18 m 51 s

Provisioning

Configuration Status	In Sync
OS Version	3.12.0JT-00461-g5be590d-dirty
Distr. ID	
Distr. Release	
JRE Version	1.7.0_65

Current view from: 12/03/2014 12:11:04 PM to: 12/03/2014 01:11:04 PM

12/02/2014 01:11:04 PM

FlexIP

- FlexIP Receive Bit Rate [bps]
- FlexIP Receive Packet Rate [pps]
- FlexIP Receive Error Rate [pps]
- FlexIP Transmit Bit Rate [bps]
- FlexIP Transmit Packet Rate [pps]
- FlexIP Transmit Error Rate [pps]

Ethernet

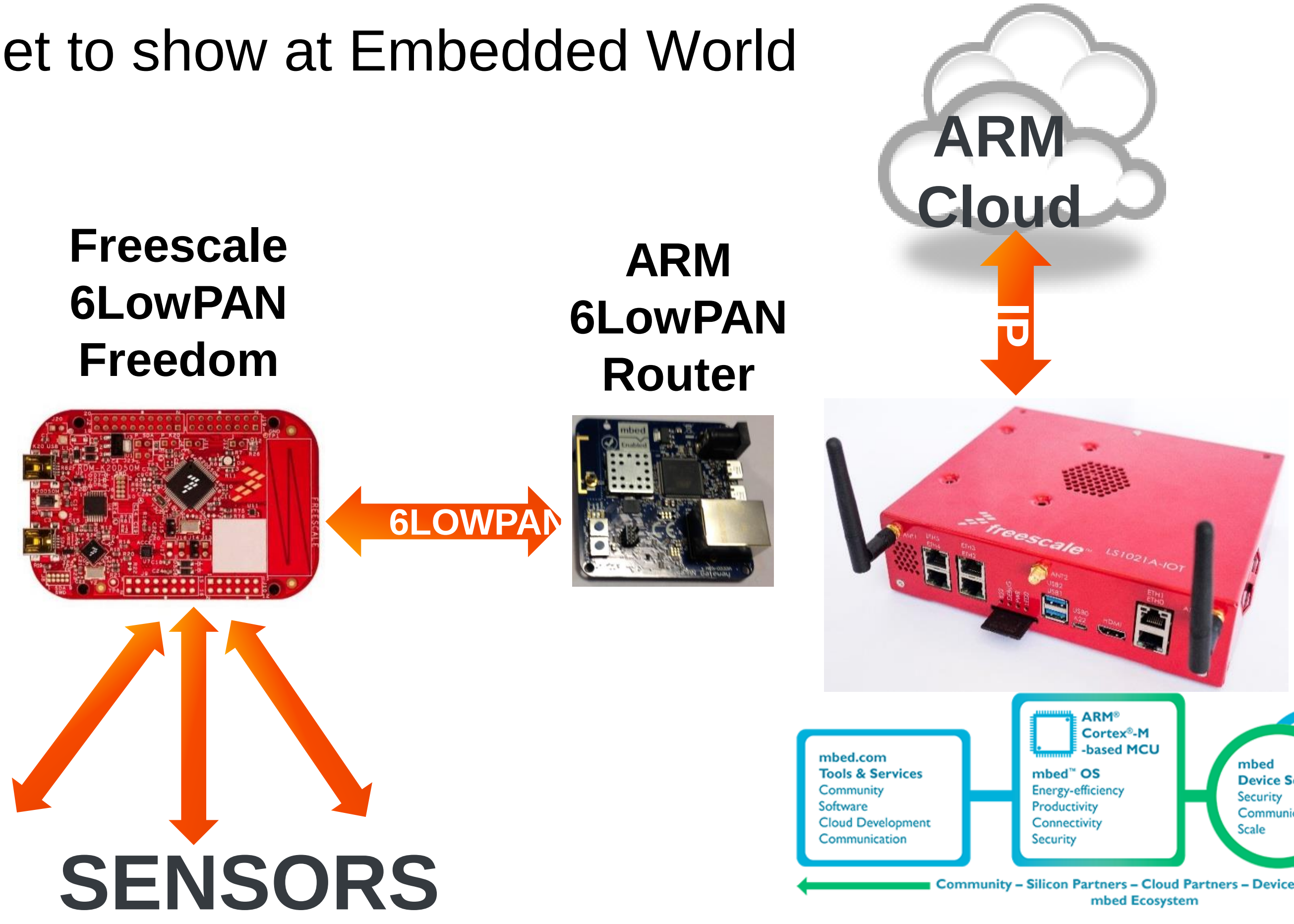
Proximetry Gateway Demo – building the demo

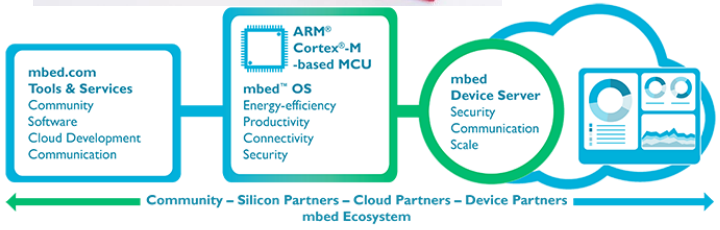
- TWR-KW24D512 and USB-KW24D512 hardware supplied by Microcontroller team with software preconfigured
- Proximetry supplied accounts for cloud interface
- LS1 software:
  - Currently using Debian file system for OpenJDK
    - Yocto support planned for end March
  - Proximetry server runs as Java application
    - passes packets to/from cloud
    - Ethernet used in current demo for cloud connectivity
- Stack from Microcontroller group runs on USB dongle
    - Ethernet tunnel runs on top of ttyACM0 between dongle and LS1 to transfer data

ARM Gateway Demo

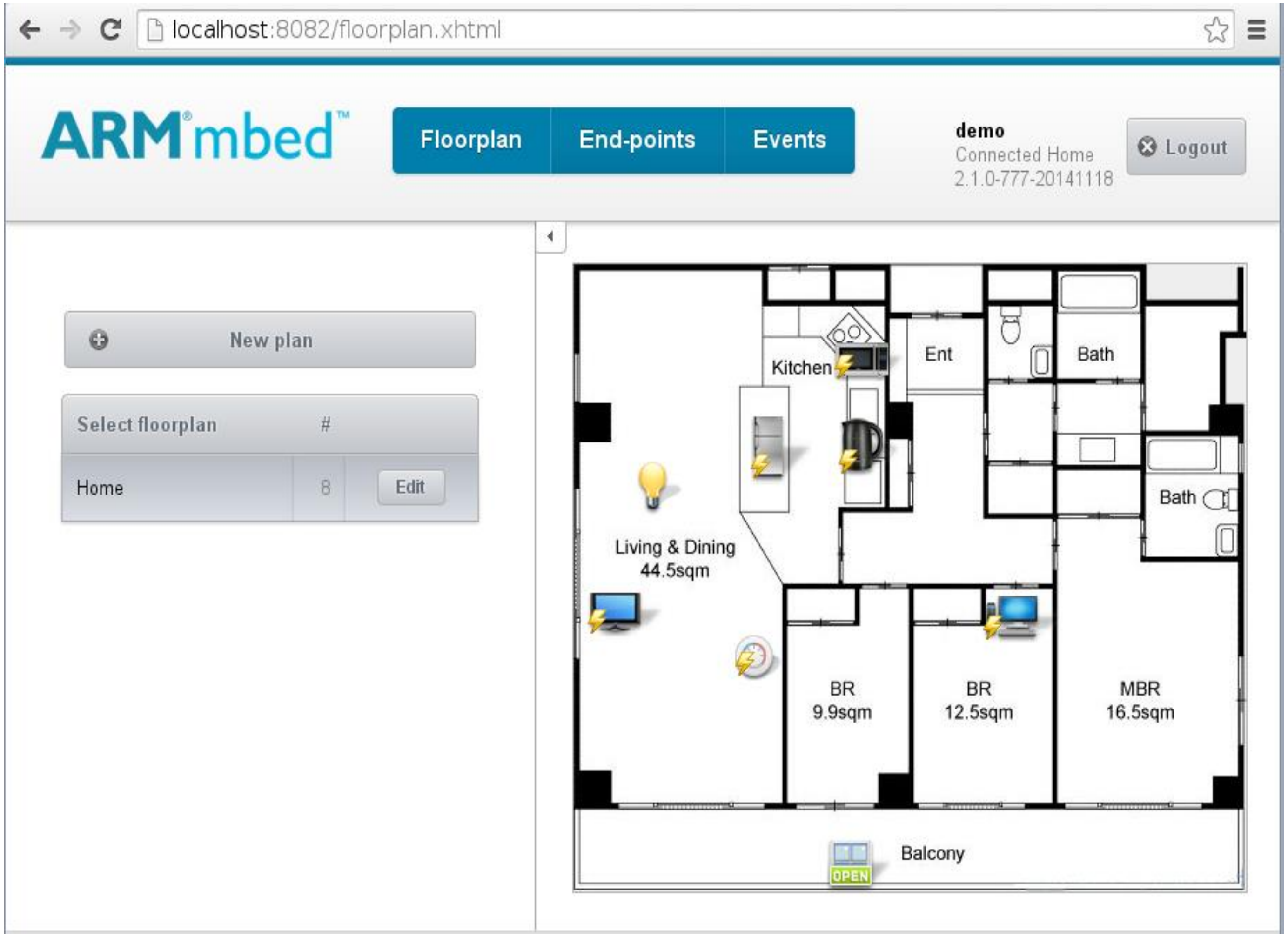
ARM 6LowPAN based IOT System Demo

- Leverage ARM 6LowPAN Demo System
  - Uses mbed Sensinode stacks and ARM Device Platform cloud
- Target to show at Embedded World





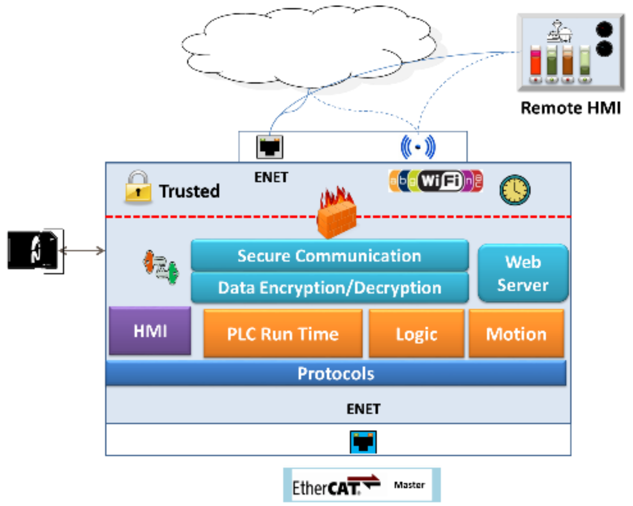
ARM Cloud Interface

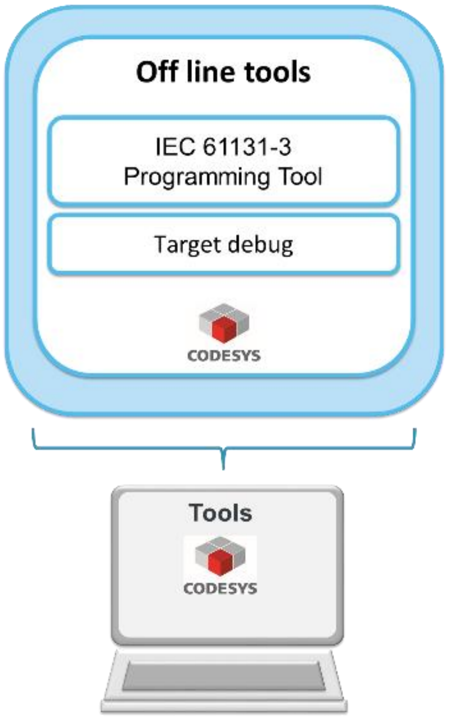


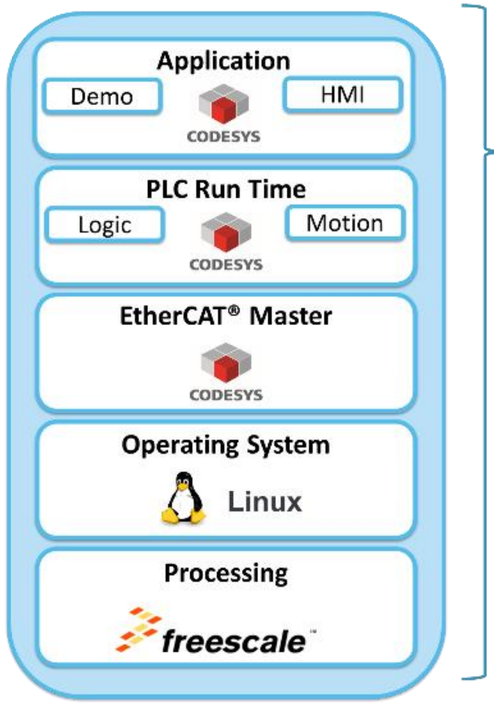
Select floorplan	#	
Home	8	Edit

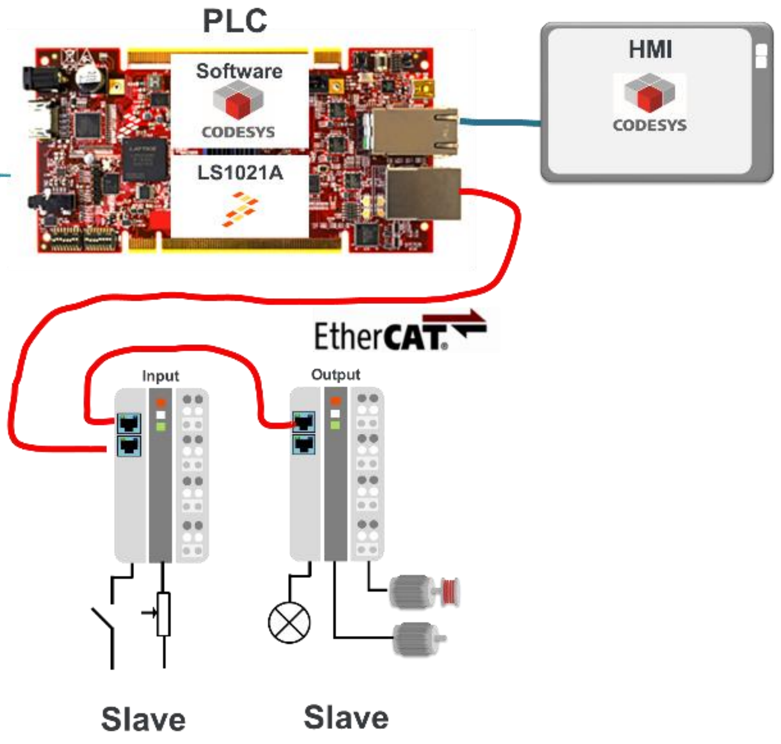
Programmable Logic Controller

Programmable Logic Controller with Web HMI and Simple Motion







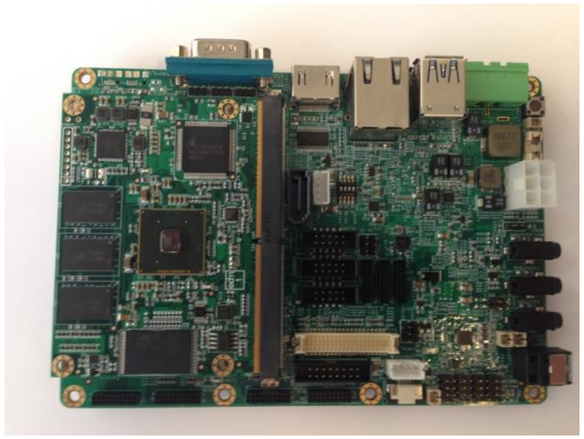


LS1 Community Kit

The LS1 community kit

<u>Objective: develop and distribute a low cost, mass market, 'self supporting' LS1 community kit</u>

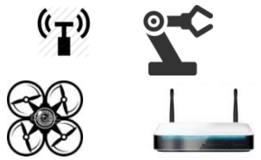
LS1 community board



Community

Linux OS

Demos & How To's



Open Source Hardware:

- LS1021A
- Display: HDMI;
- LAN: 2x 10/100/1000Mbps
- Storage:1 x SATA interface, 1 x CF slot ,1 x SPI FLASH
- I/O ports: 1 x USB3.0 ports;1 x Mini PCI-e slot
- Expansion bus: 1 x UART, 1x USB2.0 ports, 1 x CAN bus ports, +more

Differentiators: Networking throughput!

- 2 x GE's, mPCIe, SATA, ECC, higher encryption throughput

Open Source Software: Linux OS with OpenWRT

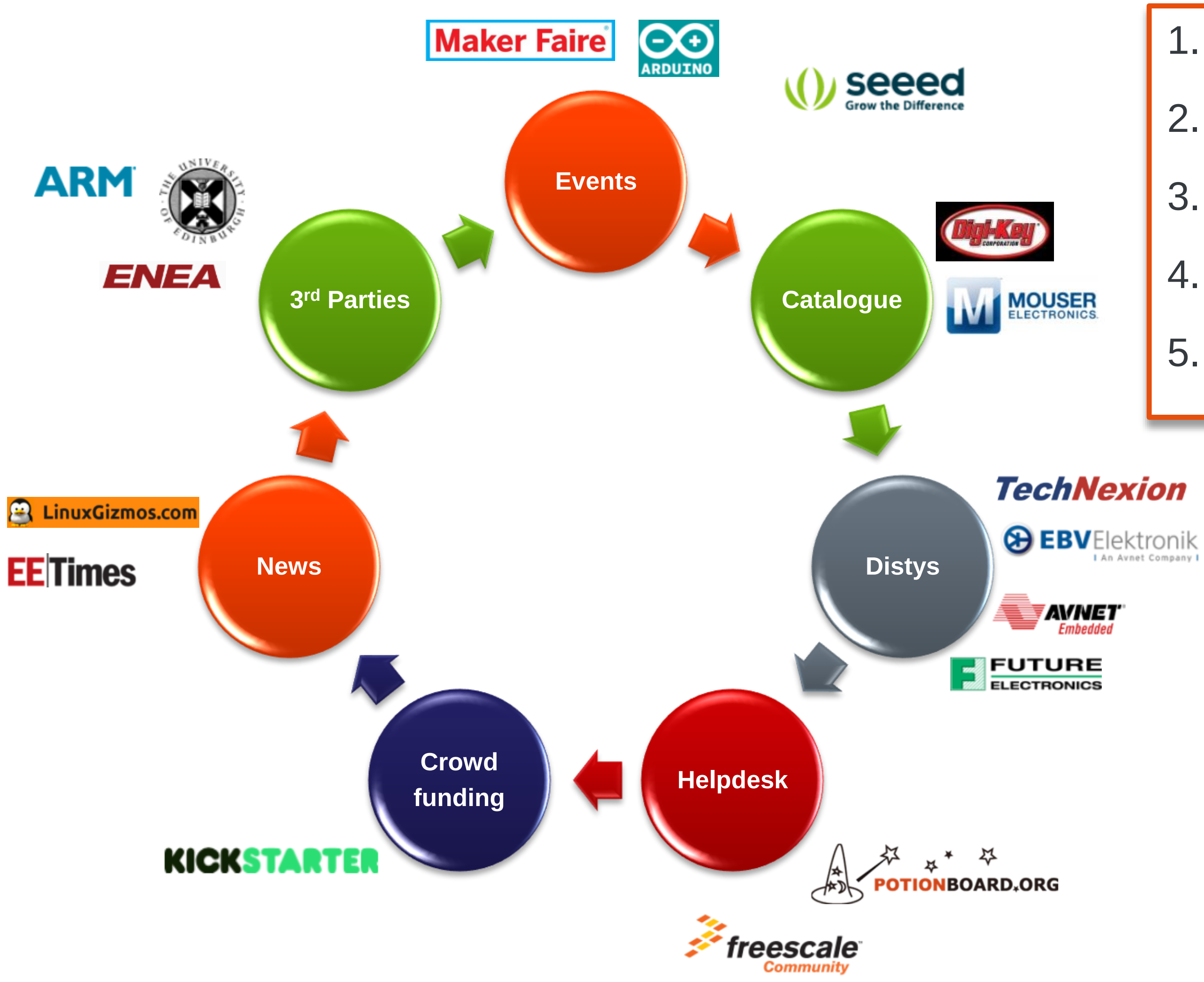
Price: circa $105USD

Community website with online Support, Demos, How-To, Set up

Accessories: WIFI/3G/4G/6LOWPAN/Zigbee/Bluetooth + more

** Not included in $105USD*

Target Communities – call to action



1. Publish **Recipes** everywhere
2. Attend Community **events**
3. Build a **Syllabus**
4. Find **Makers**!
5. Find a **Kickstarter**!

Collateral

LS1021A-IOT
Reference Materials

- Schematic : https://www.freescale.com/livelink/livelink?func=ll&objId=232616949
- BOM : https://www.freescale.com/livelink/livelink?func=ll&objId=232601994
- Layout : https://www.freescale.com/livelink/livelink?func=ll&objId=232636920
- QSG : http://cache.freescale.com/files/32bit/doc/quick_ref_guide/LS1021A-IOTGS.pdf
- LS1021A U-Boot and Linux Porting Guide: *With docs for publication*
- Powering the LS102xA with the MC34VR500 Power Management Interface Chip: *With docs for publication*

LS1021A-IOT Gateway Reference Design

- Part number: LS1021A-IOT
- Price: $429
- http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=LS1021A-IoT

freescale™

www.Freescale.com