Laboratoire d'Annecy de Physique des Particules

IPMC tutorial

IPMC Workshop

Tuesday 9th October, 2018

Fatih Bellachia

ATLAS EXPERIMENT

cnrs

UNIVERSITÉ SAVOIE MONT BLANC

- How to install/build ICARE?
- CMT
- ICARE libraries
- FRU Information and SDR
- Implementing a Software Module
- Firmware Upgrade
- Debugging with serial console

- Latest release is **ICARE-00-03-00**
- Software bundle will be available at LAPP TWIKI
- Provided as self-extractable archive
- Archive types:
  - ICARE software stack
    - ICARE_<DDMMYY>.sh
  - ICARE release
    - ICARE-<vv>-<rr>-<pp>.sh

- How to install and build
  - From scratch

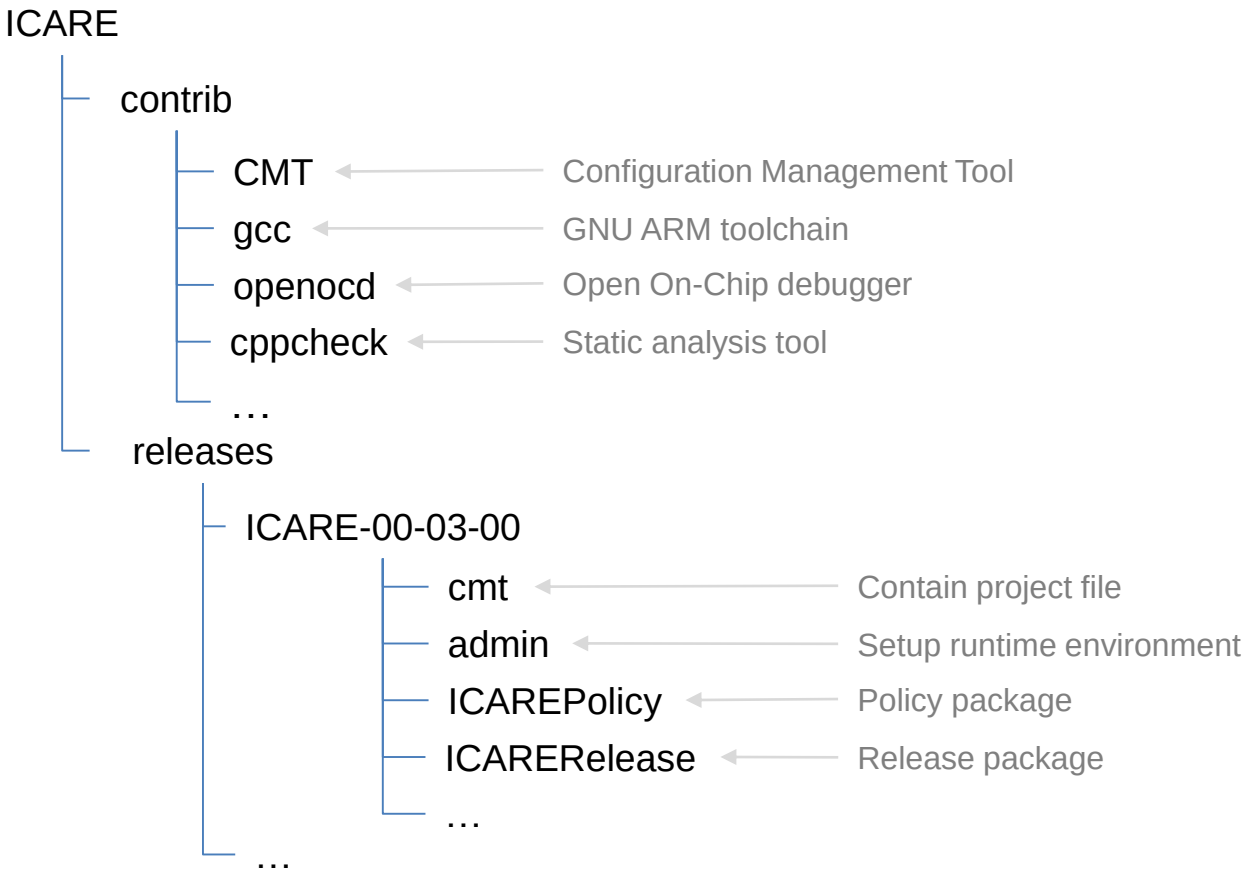
```
% cd <install area>
% <somewhere>/ICARE_<DDMMYY>.sh
```

  - For an update

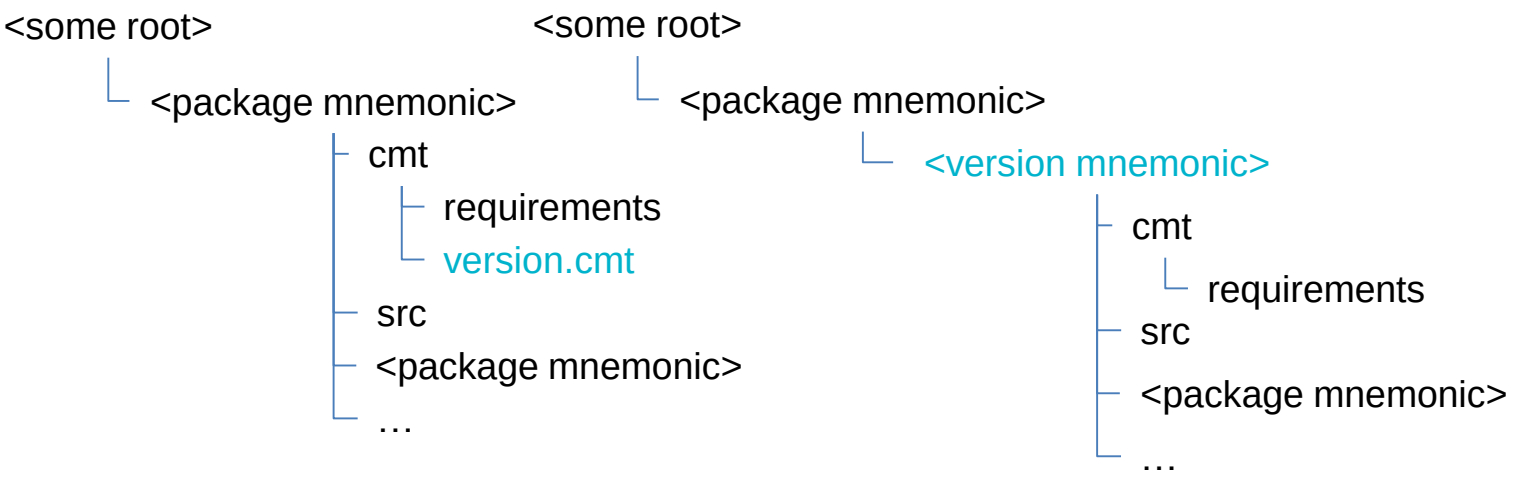
```
% cd <install area>
% <somewhere>/ICARE-<vv>-<rr>-<pp>.sh
```

The release is installed in ICARE/releases/ICARE-<vv>-<rr>-<pp>

- Directory structure



- The Configuration Management Tool (CMT)
  - Is build-automation and configuration management utility, around a packet-oriented principle.
  - It use several environment variables and text files (i.e. requirements and project.cmt) to describe the configuration requirements.
  - It deducts from the description the actual set of configuration parameters necessary for the operation of packages (i.e. to build, use or query).
- Package Directory Structures



- CMT project.cmt file

```
project ICARE-00-03-00

build_strategy no_prototypes
build_strategy rebuild_makefiles
build_strategy without_installarea

setup_strategy root
setup_strategy no_config
setup_strategy no_cleanup

structure_strategy without_version_directory
```

Project build strategy

Project setup strategy

Project structure strategy

- CMT requirements file

```
package demo

manager fatih.bellachia@lapp.in2p3.fr
author  fatih.bellachia@lapp.in2p3.fr

use libopencm3 v0r*
use support *
use rcc v0r*
use usart v0r*
use module v0r*
use blinky v0r*
use main v0r*

private

#---------------------------------------------------------------------------
# ARM applications
#---------------------------------------------------------------------------
apply_pattern arm-application name=blinky target_mcu=IPMC
apply_pattern arm-application name=blinky target_mcu=IOIF

#---------------------------------------------------------------------------
# FLASH programming
#
# Usage:
# apply_pattern flash host=<host> name=<program name> target_mcu=<IPMC|IOIF>
#                     [target_suffix=none]
#---------------------------------------------------------------------------
macro openocd_host "localhost"

apply_pattern flash host=$(openocd_host) name=blinky target_mcu=IPMC
apply_pattern flash host=$(openocd_host) name=blinky target_mcu=IOIF
```

Name

Used packages

- ARM CMT statements

  document arm-library <library name> [<constituent option …>] <source …>

  ```
  document arm-library i2c i2clib.c
  ```

  document arm-application <application name> [<constituent option …>] <source …>

  ```
  document arm-application i2c_USR_tx -group=test -s=../test/tx i2c_USR_tx.c
  ```

  document arm-object <object name> [<constituent option …>] <source …>

  ```
  document arm-object startup_stm32f4xx -group=test startup_stm32f4xx.c
  ```

- ARM CMT patterns

  apply_pattern arm-module name=<module name> target_mcu=<IPMC|IOIF> source=<source …>

  ```
  apply_pattern arm-module name=blinky_led0 target_mcu=IPMC
  source=blinky_led0.c
  ```

  apply_pattern arm-application name=<application name> [option=<option …>] target_mcu=<IPMC|IOIF>

  ```
  apply_pattern arm-application name=blinky target_mcu=IPMC
  ```

  apply_pattern flash host=<host> name=<program name> target_mcu=<IPMC|IOIF> [target_suffix=none]

  ```
  apply_pattern flash host=localhost name=blinky target_mcu=IPMC
  ```

  apply_pattern debug host=<host> name=<program name> target_mcu=<IPMC|IOIF> [target_suffix=none]

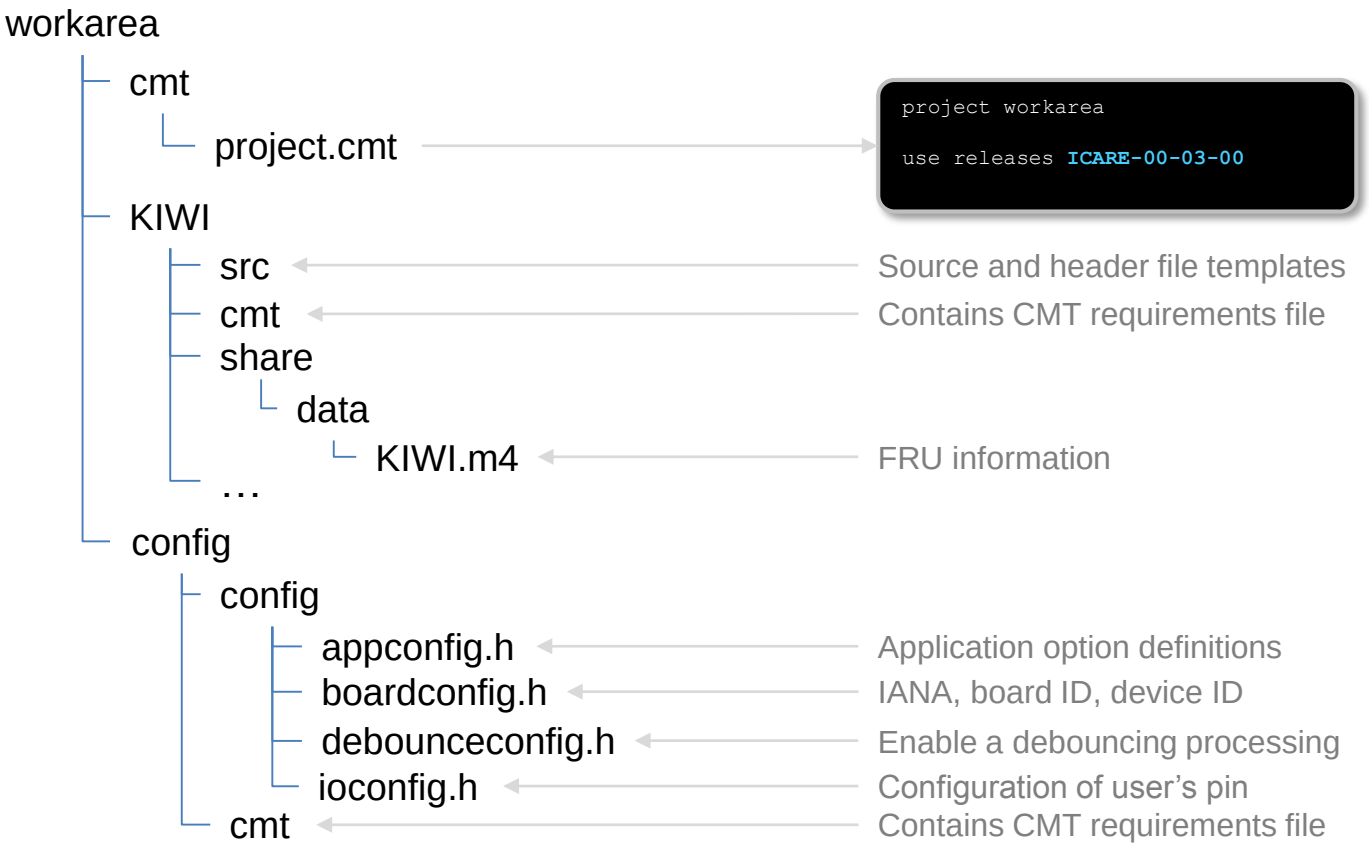
  ```
  apply_pattern debug host=localhost name=blinky target_mcu=IPMC
  ```

- Useful CMT commands
  - cmt [help] *(display help)*
  - cmt show uses *(gives the ordered and flattened set of used packages)*
  - cmt show path *(lists the effective search path for packages)*
  - cmt relocate *(generate setup and cleanup scripts)*
  - cmt broadcast -local <command> *(apply a command to the local used packages)*
- Useful Makefile targets
  - make help *(print all targets)*
  - make all *(rebuild all constituents)*
  - make clean *(remove everything that can be rebuilt)*
  - make flash_<application name> *(program MCUs)*
  - make debug_<application name> *(start INSIGHT. A Tcl/Tk-based GUI for gdb)*
  - make info *(print information about MCUs)*
  - make version *(print build date/time, name, etc…)*
  - make full_version *(make version + list of packages with their version)*

- Create your workarea

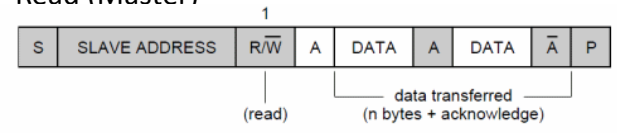
```
% source <install area>/ICARE/releases/ICARE-00-03-00/admin/cmt/setup.sh
...
% create_project_area.sh -h
usage: create_project_area.sh [-hl] -a <author> [-c] -n <host> -p <package> [-v <version>]
Create your package.
options:
        -a <author>      <author> name (email address)
        -c               Add Carrier Management Controller.
        -h               Print this help
        -l               List existing package templates
        -n <host>        <host> name to run the OpenOCD server on
        -p <package>     <package> name
        -v <version>     <version> number of package (default: v0r1)

% mkdir workarea
% cd workarea
% create_project_area.sh -a fatih.bellachia@lapp.in2p3.fr -n localhost -p KIWI -v v0r1
Project created by bellac Oct 05, 2018 at 01:14:57 PM

Your parameters
---------------

package:  KIWI
version:  v0r1
author:   fatih.bellachia@lapp.in2p3.fr
hostname: localhost
F/W type: IPMC

Do you want to continue? [yes, no, abort] (y) :
...
```

- Directory structure

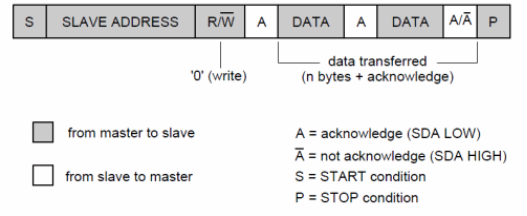


- i2c
- iopin
- Channel
- Message
- MessageQueue
- ekeying
- resourceBroker
- nsBrocker

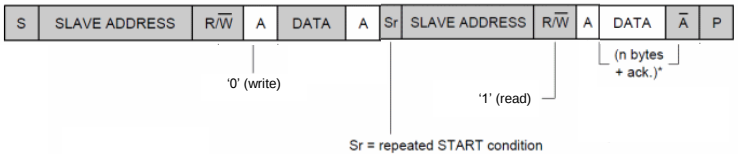
- I2C peripheral
  - Speed mode (100 kbit/s)
  - 7-bit addressing mode
  - Interrupt driven
  - **Master** and slave access mode
    - Read (Master)



    - Write (Master)



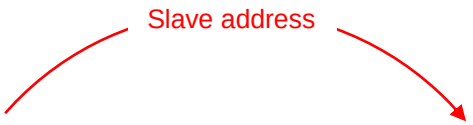
    - Write and Read (Master)



- I2C peripheral API

int I2C_Read(uint32_t uiI2Cx, I2CXferMode_t eMode, uint8_t *pucBuffer, uint32_t uiCount, ...)

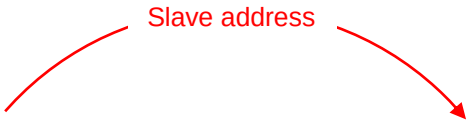
- Master READ




```
iRtn = I2C_Read(I2C_SENSOR, I2C_MASTER, &ucBuffer, uiBytes, 0x43);
```

int I2C_Write(uint32_t uiI2Cx, I2CXferMode_t eMode, const uint8_t *pucBuffer, uint32_t uiCount, ...)

- Master WRITE




```
iRtn = I2C_Write(I2C_SENSOR, I2C_MASTER, &ucBuffer, uiBytes, 0x43);
```

int I2C_MasterWriteRead(uint32_t uiI2Cx, uint8_t slave, uint8_t *pucWrBuf, uint32_t uiWrCount, uint8_t *pucRdBuf, uint32_t uiRdCount)

- Master WRITE and READ

```
iRtn = I2C_MasterWriteRead(I2C_SENSOR, 0x43, &aucWrBuf[0], uiWrBytes,
                           &aucRdBuf[0], uiRdBytes);
```

The iopin library relies on the gpio library, it allows to get rid of the real state of the signals by taking into account only the polarity of the inputs and outputs chosen by the user.

	Input	Output
States	IO_PIN_IN_INACTIVE IO_PIN_IN_ACTIVE IO_PIN_IN_ACTIVE_TRANSITION IO_PIN_IN_INACTIVE_TRANSISTION	IO_PIN_OUT_NEGATE IO_PIN_OUT_ASSERT
Polarities	IO_PIN_ACTIVE_HIGH IO_PIN_ACTIVE_LOW	
	IO_PIN_NORMALLY_OPENED *IO_PIN_NORMALLY_CLOSED*	

- iopin defintion *(see config/ioconfig.h)*

```
IO_PIN_DEFINE(GPIO_USER_IO_1, IO_PIN_ACTIVE_HIGH)
IO_PIN_DEFINE(GPIO_HANDLE_SWITCH, IO_PIN_NORMALLY_CLOSED)
```

- iopin API

  bool IOPinOpen(ioPin_t uiIoPin)

```
iRtn = IOPinOpen(PIN_GPIO_USER_IO_1);
```

  bool IOPinClose(ioPin_t uiIoPin)

```
iRtn = IOPinOpen(PIN_GPIO_USER_IO_1);
```

  ioPinInState_t IOPinInGetState(ioPin_t uiIoPin)

```
If(IOPinInGetState(PIN_GPIO_USER_IO_1) == IO_PIN_IN_ACTIVE_TRANSITION) {
 …
}
```

  ioPinOutState_t IOPinOutGetState(ioPin_t uiIoPin)

```
If(IOPinOutGetState(PIN_GPIO_USER_IO_12) == IO_PIN_OUT_NEGATE) {
 …
}
```

  void IOPinOutAssert(ioPin_t uiIoPin)

```
IOPinOutAssert(PIN_GPIO_IPM_IO_5);
```

  void IOPinOutNegate(ioPin_t uiIoPin)

```
IOPinOutNegate(PIN_GPIO_IPM_IO_5);
```

The channel library is an abstract layer for controlling and communicating with physical interfaces

- channel Identifiers

Channel ID	IPMI-channel #
CHANNEL_IPMB0	0x0
CHANNEL_CONSOLE	0x1
CHANNEL_I2C_MGT	0x2
CHANNEL_I2C_SENSOR	0x3
CHANNEL_I2C_USER_IO	0x4
CHANNEL_I2C_IPM_IO	0x5
CHANNEL_ETHERNET	0x6
CHANNEL_IPMBL	0x7
CHANNEL_MAILBOX	0x8
CHANNEL_OEM_CH1	0x9
CHANNEL_OEM_CH2	0xA
CHANNEL_OEM_CH3	0xB
	0xC
	0xD
	0xE
CHANNEL_IMC	0xF

- channel callbacks

```
typedef bool(*ChannelCallback_t)(void *pvArg, struct Message
*pstMsg, uint32_t status);
…
typedef struct __attribute__ ((packed)) ChannelCb {
  ChannelCallback_t fnRoutine;
  void *pvContext;
} ChannelCb_t;
…
typedef struct __attribute__ ((packed)) ChannelUserCallbacks {
  ChannelCb_t stError;    ///< callback for handling error
  ChannelCb_t stTimeout;  ///< callback for handling timeout
  ChannelCb_t stComplete; ///< callback for completion
  ChannelCb_t stClose;    ///< callback for handling close
  SLIST_ENTRY(ChannelUserCallbacks) entries;
} ChannelUserCallbacks_t;
```

- channel API

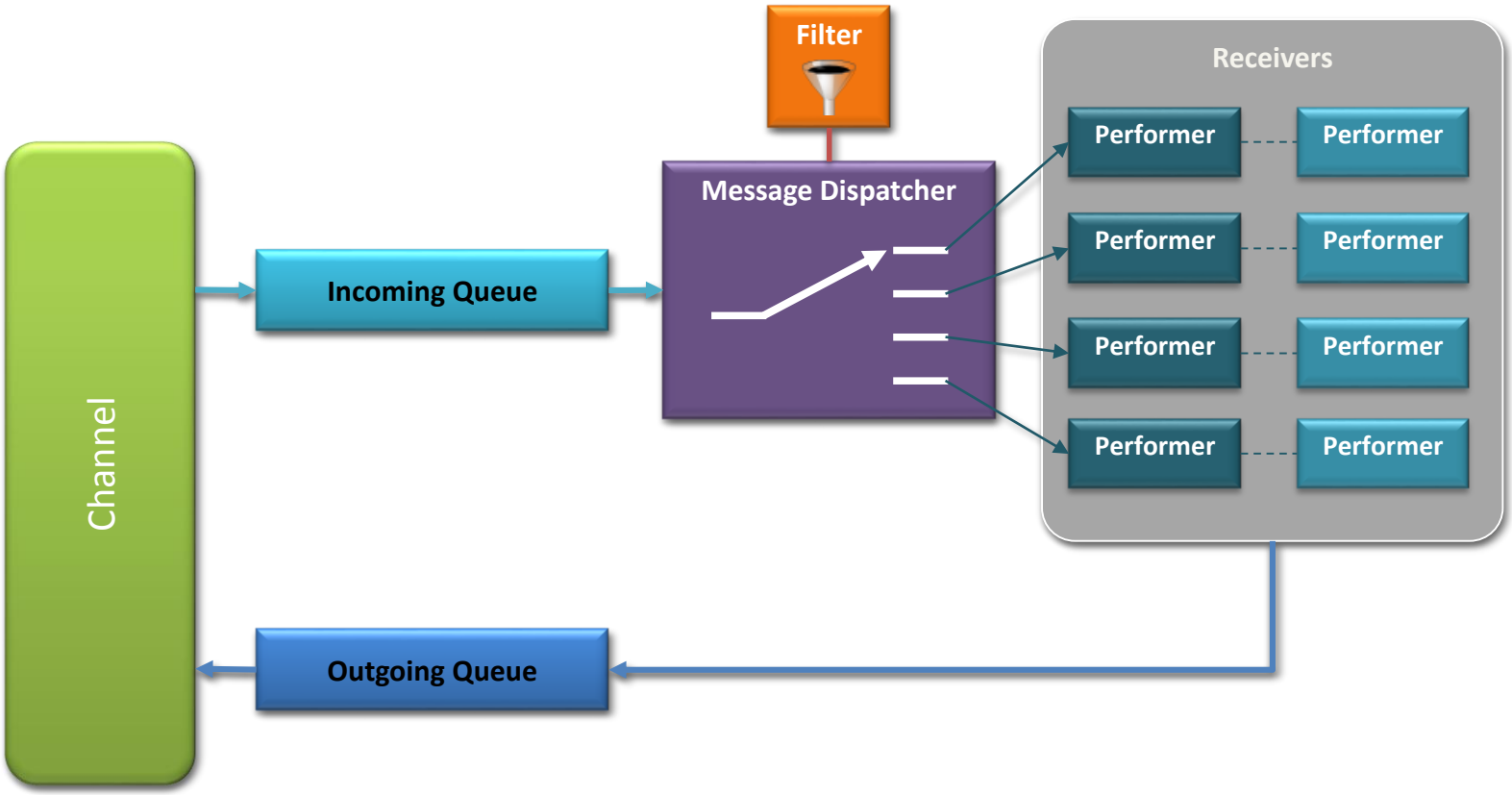
HANDLE ChannelRegisterUserCallbacks(uint8_t ucChannelId, const ChannelUserCallbacks_t* callbacks)

```
ChannelUserCallbacks_t ucb;
memset(&ucb, 0, sizeof(ucb));
ucb.stError.fnRoutine    = ErrorCallback;
ucb.stTimeout.fnRoutine  = TimeoutCallback;
ucb.stComplete.fnRoutine = CompleteCallback;
ucb.stClose.fnRoutine    = CloseCallback;
userCallbacks = ChannelRegisterUserCallbacks(CHANNEL_ETHERNET, &ucb);
```

bool ChannelUnregisterUserCallbacks(uint8_t ucChannelId, HANDLE handle)

```
bool bRtn = ChannelUnregisterUserCallbacks(CHANNEL_ETHERNET,
                                           userCallbacks);
```

The messaging is based on an asynchronous Request-Response model.



- messageQueue API

Message_t *MessageCreate(uint8_t ucChannel, uint32_t uiSize, uint8_t *pucBuffer)

```
Message_t *pMsg = MessageCreate(CHANNEL_IPMB0, uiLength, pucBuffer);
```

Message_t *MessageCreateWithNoCopy(uint8_t ucChannel, uint32_t uiSize, uint8_t *pucBuffer)

```
Message_t *pstMsg = MessageCreateWithNoCopy(CHANNEL_ETHERNET, uiLen,
                                            (uint8_t *)pucBuf);
```

Message_t *MessageClone(Message_t *pstMsg)

```
Message_t *pMsg = MessageClone(pReqMsg);
```

Message_t *MessageDestroy(Message_t *pstMsg)

```
pMsg = MessageDestroy(pRespMsg);
```

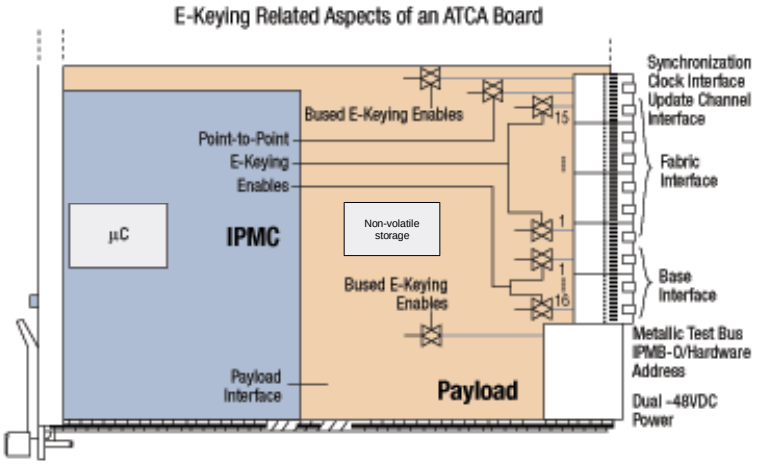
bool MessageQueueOutPut(Message_t *pstMsg)

```
if (MessageQueueOutPut(pReqMsg) == false) {
  …
```

bool MessageQueueInPut(Message_t *pstMsg)

```
if (MessageQueueInPut(pRespMsg) == false) {
  …
```

The point-to-point E-Keying is a protocol used to describe the compatibility between the Base Interface, Fabric Interface and Update Channel Interface connections of Front Boards.



The board's non-volatile storage include FRU information with E-Keying data (i.e. link descriptor). The Shelf Manager use these information to managed the P2P connections to the backplane.

- Link Descriptor

  32-bit word which identifies link and an associated protocol.

Bit offset	Description
31:24	*Link Grouping ID.* Indicates whether the Ports of this Channel are operated together with Ports in other Channels. A value of 0 always indicates a Single-Channel Link. A common, non-zero *Link Grouping ID* in multiple *Link Descriptors* indicates that the Ports covered by those *Link Descriptors* must be operated together. A unique non-zero *Link Grouping ID* also indicates Single-Channel Link.
23:20	*Link Type Extension.* Identifies the subset of a subsidiary specification that is implemented and is defined entirely by the subsidiary specification identified in the *Link Type* field.
19:12	*Link Type.* Identifies the PICMG® 3.x subsidiary specification that governs this description or identifies the description as proprietary; see Table 3-52, "Link Type."
11:0	*Link Designator.* Identifies the Interface Channel and the Ports within the Channel that are being described; see Table 3-51, "Link Designator."

- ekeying API

bool EKRegisterFunc(linkDescriptor descr, EKFunction_t function)

```
#define BASE_IF_CH1_PORT0 0x00001101
…
bool SetPortStateCallback(linkDescriptor linkInfo, char state)
{
  …
  // Change state of Link
  if (state == 1) { // Enable port
    …
  }
  else { // Disable port
    …
  }
  …
}
…
bRtn &= EKRegisterFunc(BASE_IF_CH1_PORT0, SetPortStateCallback);
```

The resource broker allows to register and manage the sensors according to the channel number, I2C address and sensor ID. The user provides callback functions to initialize, to write and to read the sensors.

When the IPMC receives the « Get Sensor Reading » request based on the information provided by the Shelf Manager, it performs the corresponding reading function.

- Resource Descriptor

```
typedef bool (*RBInitFunction_t)(struct RBResource *pstResource);
typedef bool (*RBReadFunction_t)(struct RBResource *pstResource, uint8_t *pucData);
typedef int (*RBWriteFunction_t)(struct RBResource *pstResource, void *pvBuffer, uint32_t uiLength);

typedef struct RBResource {
  char szName[RESOURCE_NAME_SIZE + 1];
  uint8_t ucChannelId;
  uint8_t ucAddress;
  uint8_t ucIdentifier;
  bool bInitialized;
  RBInitFunction_t fnInit;
  RBReadFunction_t fnRead;
  RBWriteFunction_t fnWrite;
  SLIST_ENTRY(RBResource) entries;
} RBResource_t;
```

Sensor name

Key

Callback function pointers

- resourceBroker API

bool ResourceBrokerAddResource(char *pszName, RBResource_t *pstResource)

```
stResource.ucChannelId  = CHANNEL_I2C_SENSOR;
stResource.ucAddress    = 0x49;
stResource.ucIdentifier = 35;
stResource.fnInit       = TMP102_InitCallback;
stResource.fnRead       = TMP102_ReadCallback;
stResource.fnWrite      = NULL;

if (ResourceBrokerAddResource("TMP102 Core Temp", &stResource) == false)
  return false;
```

RBResource_t* ResourceBrokerFindResource(uint8_t ucChannelId, uint8_t ucAddress, uint8_t ucIdentifier)

```
RBResource_t* pstResource = ResourceBrokerFindResource(CHANNEL_I2C_SENSOR,
                                                       0x49, 35);
```

RBResource_t* ResourceBrokerFindResourceByName(char *pszName)

```
RBResource_t* pstResource = ResourceBrokerFindByName("TMP102 Core Temp");
```

- resourceBroker API

bool ResourceBrokerRemoveResource(RBResource_t *pstResource)

```
bool bRtn = ResourceBrokerRemoveResource(&stResource);
```

bool ResourceBrokerRemoveResourceByName(char *pszName)

```
bool bRtn = ResourceBrokerRemoveResourceByName("TMP102 Core Temp");
```

void ResourceBrokerDumpResources(const char *szPrompt)

```
ResourceBrokerDumpResources("DEBUG>");
```

The network service broker allows to register and manage the Ethernet services according to the name, the protocol and the port. In case of response time issue the user can provides callback function to receive and process incoming data.

- nsBroker API

bool NSBrokerAddService(const char *szName, uint16_t usPort, protocol_t eProtocol, HANDLE userCallbacks)

```
ChannelUserCallbacks_t ucb;
memset(&ucb, 0, sizeof(ucb));
ucb.stError.fnRoutine = ErrorCallback;
ucb.stTimeout.fnRoutine = TimeoutCallback;
ucb.stComplete.fnRoutine = CompleteCallback;
ucb.stClose.fnRoutine = CloseCallback;
userCallbacks = ChannelRegisterUserCallbacks(CHANNEL_ETHERNET, &ucb);
…
bool bRtn = NSBrokerAddService("TCP Echo", 7, NS_TCP, userCallbacks);
…
s_hEcho_Performer = MessageDispatcherRegisterCallbacks(CHANNEL_ETHERNET,
                        Echo_Performer,
                        (MsgDispatcherCallback_t)NSBrokerCheckService,
                        pstNetService);
```

- nsBroker API

bool NSBrokerRemoveService(netService_t* pstNetService)

```
netService_t stNetService;
…
bool bRtn = NSBrokerRemoveService(&nsNetService);
```

bool NSBrokerRemoveServiceByName(const char *szName, protocol_t eProtocol)

```
bool bRtn = NSBrokerRemoveServiceByName("TCP Echo", NS_TCP);
```

bool NSBrokerRegisterCallback(netService_t* pstNetService, netServiceCallback_t fnCallback, void *pvContext)

```
…
static bool ETH_Performer(void *pvContext, Message_t *pstMsg)
{
  …
}
…
bool bRtn = NSBrokerRegisterCallback(&stNetService, ETH_Performer,
                                     &stNetService);
```

ATCA supports the storage of Field Replaceable Unit (FRU) information for various modules in the system. The FRU data contains information like manufacturer, serial numbers, part numbers, models and inventory numbers and E-Keying data.

- FRU Information layout

- Example of definitions in M4 file

```
dnl
dnl IPMI_BOARD_INFO_AREA
dnl
IPMI_BOARD_MANUFACTURER(FRU_ID, LAPP/CNRS)
IPMI_BOARD_PRODUCT(FRU_ID, KIWI)
IPMI_BOARD_PART_NUMBER(FRU_ID, KIWI)
IPMI_BOARD_MFG_DATE(FRU_ID, 0x20, 0xCA, 0x8D)
IPMI_BOARD_SERIAL_NUMBER(FRU_ID, 1234DK001)
IPMI_BOARD_FRU_FILE_ID(FRU_ID, fru_data.c)
dnl
dnl IPMI_PRODUCT_INFO_AREA
dnl
IPMI_PRODUCT_MANUFACTURER(FRU_ID, LAPP/CNRS)
IPMI_PRODUCT_PART_NUMBER(FRU_ID, KIWI)
IPMI_PRODUCT_PRODUCT(FRU_ID, LAPP-IPMC-DEVKIT)
IPMI_PRODUCT_VERSION(FRU_ID, 01)
IPMI_PRODUCT_SERIAL_NUMBER(FRU_ID, 1234DK001)
IPMI_PRODUCT_ASSET_TAG(FRU_ID)
IPMI_PRODUCT_FRU_FILE_ID(FRU_ID)
dnl
dnl PICMG_BOARD_P2P_CONNECTIVITY_RECORD
dnl
PICMG_LNK_DESC_ENTRY(FRU_ID, 0x1, PICMG_LNK_DSG_IF_BASE, PICMG_LNK_DSG_PORT_INCLUDED,
PICMG_LNK_DSG_PORT_EXCLUDED, PICMG_LNK_DSG_PORT_EXCLUDED, PICMG_LNK_DSG_PORT_EXCLUDED,
PICMG_LNK_TYPE_BASE_IF, PICMG_LNK_EXT_BASE_T_LINK, 0x0)
```

- Sensor Data Records are records that contain information about the type and number of sensors. A sensor data record therefore describes a specific sensor.
- The general Sensor Data Record format consists of three major components
  - Record Header

    Record ID: A value that's used for accessing Sensor Data Records

    SDR Version: The version number of the SDR specification. Used in conjunction with Record Type

    Record Type: A number representing the type of the record

    Record Length: Number of bytes of data following the Record Length field.
  - Record Key Fields

    The Record "Key" Fields are a set of fields that together are unique amongst instances of a given record type, these fields specify the location (e.g. slave address, LUN, and **channel ID**) and **sensor number** of the sensor.
  - Record Body

    The remaining Record Type specific information for the particular sensor data record.

- The src/sdr_data.c file contains mandatory SDR records and some sensor record templates. From the templates the user can modify or add other records for describing the properties of sensors.
- Fields to be completed
  - Sensor Class: Threshold or Discrete
  - Type: e.g. Temperature, Voltage, Current
  - Range, Resolution and Tolerance
  - Units: e.g. degrees C, Volts, Amps
  - Thresholds: Upper and Lower, Non-Critical, Critical, Non-Recoverable
  - Conversion parameters
    - $y = L[(Mx + (B * 10^{K1})) * 10^{K2}]$ units

x	Raw reading
y	Converted reading
L[ ]	Linearization function specified by 'linearization type'. This function is 'null' (y = f(x) = x) if the sensor is linear.
M	Signed integer constant multiplier
B	Signed additive 'offset'
K1	Signed Exponent. Sets 'decimal point' location for B. This is called the 'B' exponent in the SDRs.
K2	Signed Result Exponent. Sets 'decimal point' location for the result before the linearization function is applied. this is called the 'R' exponent in SDRs. Linear and Linearized readings have constant accuracy, tolerance, M, and B factors regardless of the reading.

- Channel ID *(see channel/channel.h)*
  - CHANNEL_I2C_MGT *(managed by IPMC)*
  - CHANNEL_I2C_SENSOR *(managed by IPMC)*
  - CHANNEL_I2C_USER_IO
  - CHANNEL_I2C_IPM_IO

Exclusive

    - <Full|Compact> Sensor Record

      set **BYTE 7** [4:7] channel_num

- Sensor Number
  - Unique number identifying the sensor *(see src/cfg_data.h)*
    - <Full|Compact> Sensor Record

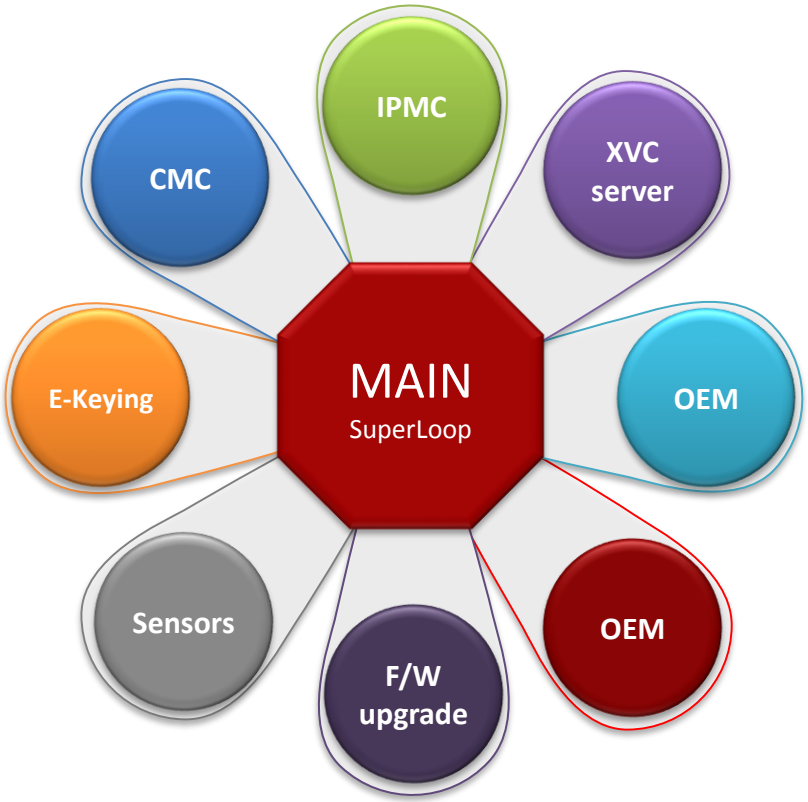
      set **BYTE 8** [0:7] sensor_number

- I2C address of sensor
  - Compact Sensor Record

    set **BYTE 31** [0:7] OEM - Reserved for OEM use
  - Full Sensor Record

    set **BYTE 47** [0:7] OEM - Reserved for OEM use

The software «module» concept allows user to extend functionalities of IPMC without modifying existing code.



- Module manifest declaration *(see module/module.h)*
  - User need to provide 3 entry point functions and information.
    - Initialization function
    - Cleanup function
    - Processing function

MODULE_BEGIN_DECL(<name>, <order>)
  MODULE_NAME("<module name>")
  MODULE_AUTHOR("<author>")
  MODULE_INIT(<Init function>)
  MODULE_CLEANUP(<Clean function>)
  MODULE_PROCESS(<Process function>, <arg>)
  MODULE_PERIODIC_PROCESS(<Process function>, <arg>, <period>)
MODULE_END_DECL

Exclusive

```
…
MODULE_BEGIN_DECL(toggle_gpio, 1)
  MODULE_NAME("Toggle user GPIO #1")
  MODULE_AUTHOR("fatih.bellachia@lapp.in2p3.fr")
  MODULE_INIT(ToggleGPIOInit)
  MODULE_CLEANUP(ToggleGPIOCleanup)
  MODULE_PERIODIC_PROCESS(ToggleGPIOProcess, NULL, 5000)
MODULE_END_DECL
```

- Register your E-Keying data

```
/*-------------------------------------------------------------------------*/
bool set_port_state_callback(linkDescriptor linkInfo, char state)
/*-------------------------------------------------------------------------*/
{
  if (linkInfo == 0x00001101)
    do something…
  else if (linkInfo == 0x00001102)
    do something else…

  …

  return true;
}

/*-------------------------------------------------------------------------*/
bool P2PEKeyingInit(void)
/*-------------------------------------------------------------------------*/
{
  …
  EKRegisterFunc(0x00001101, set_port_state_callback);
  EKRegisterFunc(0x00001102, set_port_state_callback);

  …

  return true;
}
```

Base I/F - channel 1

Base I/F - channel 2

- Provide Init callback

```
/*---------------------------------------------------------------------------*/
static bool TMP102_InitCallback(struct RBResource *pstResource)
/*---------------------------------------------------------------------------*/
{
  // Sanity check
  if (pstResource == NULL)
    return false;

  // Check I2C address range (see tmp102/tmp102.h)
  switch (pstResource->ucAddress) {
    case TMP102_SLAVE_ADDR_GND_PIN:
    case TMP102_SLAVE_ADDR_VDD_PIN:
    case TMP102_SLAVE_ADDR_SDA_PIN:
    case TMP102_SLAVE_ADDR_SCL_PIN:
      break;
    default:
      return false;
  }

  // Get I2C channel
  uint32_t uiI2Cx = auiChannel2Bus[pstResource->ucChannelId];

  // Sanity check
  if (uiI2Cx == 0xFFFFFFFF)
    return false;

  return (tmp102Config(uiI2Cx, pstResource->ucAddress) != -1);
}
```

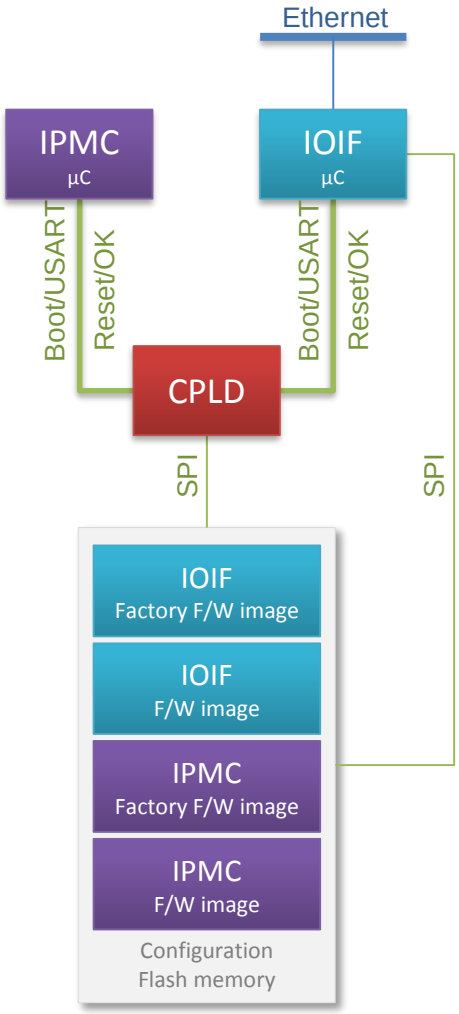
- Provide Read callback

```
/*-------------------------------------------------------------------------*/
static bool TMP102_ReadCallback(struct RBResource *pstResource, uint8_t *pucData)
/*-------------------------------------------------------------------------*/
{
  // Sanity check
  if ((pstResource == NULL) || (pucData == NULL))
    return false;

  // Check I2C address range (see tmp102/tmp102.h)
  switch (pstResource->ucAddress) {
    case TMP102_SLAVE_ADDR_GND_PIN:
    case TMP102_SLAVE_ADDR_VDD_PIN:
    case TMP102_SLAVE_ADDR_SDA_PIN:
    case TMP102_SLAVE_ADDR_SCL_PIN:
      break;
    default:
      return false;
  }

  // Get I2C channel
  uint32_t uiI2Cx = auiChannel2Bus[pstResource->ucChannelId];

  // Sanity check
  if (uiI2Cx == 0xFFFFFFFF)
    return false;

  int16_t sValue = 0;

  // Read temperature
  if (tmp102Read(uiI2Cx, pstResource->ucAddress, &sValue) == false)
    return false;

  *pucData = (sValue & 0x0FFF) >> 3;

  return true;
}
```

- Register your sensors

```
/*-------------------------------------------------------------------------*/
bool SensorsInit()
/*-------------------------------------------------------------------------*/
{
  RBResource_t stResource;

  // TMP102 Temperature
  stResource.ucChannelId  = CHANNEL_I2C_SENSOR;       // defined in channel/channel.h
  stResource.ucAddress    = SDR_TMP102_I2C_ADDR;      // defined in KIWI/src/cfg_data.h
  stResource.ucIdentifier = SDR_NUM_TMP102_CORE_TEMP; // defined in KIWI/src/cfg_data.h
  stResource.fnInit       = TMP102_InitCallback;
  stResource.fnRead       = TMP102_ReadCallback;
  stResource.fnWrite      = NULL;

  if (ResourceBrokerAddResource("TMP102 Core Temp", &stResource) == false)
    return false;

  // ADS1115 3.3V Voltage
  stResource.ucChannelId  = CHANNEL_I2C_SENSOR;
  stResource.ucAddress    = SDR_ADS1115_I2C_ADDR;
  stResource.ucIdentifier = SDR_NUM_ADS1115_3V3_VOLTAGE;
  stResource.fnInit       = ADS1115_InitCallback;
  stResource.fnRead       = ADS1115_ReadCallback;
  stResource.fnWrite      = NULL;

  if (ResourceBrokerAddResource("ADS1115 3.3V Voltage", &stResource) == false)
    return false;

  // Dump resources
  ResourceBrokerDumpResources("DEBUG> [SensorsInit]");

  return true;
}
```

Channel #
I2C address
Unique Sensor ID
User's callbacks



- Factory Firmware will be initially stored in both the Configuration flash memory and micro-controllers internal flash memory.
- The factory firmware always remains available in the IPMC flash memory.
- The micro-controllers will revert to it after 3 failing consecutive attempts of upgrading the internal flash memory.
- Firmware upgrade requires a new firmware in the IPMC CPLD.
  - Version 5.2 is available from the ICARE twiki page.
  - Requires the USB-JTAG Adaptor and openOCD tool for upgrading.
  - For those requiring it, IPMCs will be exchanged if you do not want to do the update yourselves.

In case of fault exception the debug library dump on serial console the stack trace and Cortex-M4 System Control Block registers which allowing the user to determine which cause raised an exception without debugger tool.

- Screenshot

```
[FAULT Exception]
Stack:
  R0       = 0000000a
  R1       = 0000000a
  R2       = 00000038
  R3       = deadface
  R12      = 00000000
  LR [R14] = 080049d5  subroutine call return address
  PC [R15] = 08004922  program counter
  XPSR     = 61000000

Fault Reports:
  Memory Manage Faults:
      MM_FAULT_ADDR:  0xdeadface
      MM_FAULT_STAT:  0x00
      [ ] IACCVIOL    [ ] MUNSTKERR
      [ ] DACCVIOL    [ ] MSTKERR
      [ ] MMARVALID
  Bus Faults:
      BUS_FAULT_ADDR:  0xdeadface
      BUS_FAULT_STAT:  0x82
      [ ] IBUSERR      [ ] UNSTKERR
      [x] PRECISERR    [ ] STKERR
      [ ] IMPRECISERR [x] BFARVALID
  Usage Faults:
      USG_FAULT_STAT:  0x0000
      [ ] UNDEFINSTR  [ ] NOCP
      [ ] INVSTATE    [ ] UNALIGNED
      [ ] INVPC       [ ] DIVBYZERO
  Hard Faults:
      HARD_FAULT_STAT:  0x40000000
      [ ] VECTTBL     [ ] DEBUGEVT
      [x] FORCED
  Debug Faults:
      DBG_FAULT_STAT:  0x00000000
      [ ] HALTED      [ ] VCATCH
      [ ] BKPT        [ ] EXTERNAL
      [ ] DWTTRAP
```

- Find the line with Program Counter (R15)

```
[bellac@lappc-f533] /tmp/workarea/KIWI/cmt> arm-none-eabi-addr2line -e ../arm-gcc47-dbg/bmc_IOIF.elf 08004922
/tmp/workarea/KIWI/cmt/../src/toggle_gpio.c:87

 66 /** @defgroup demo module
 67   * @{
 68   */
 69
 70 /* Private typedef -----------------------------------------------------------*/
 71 /* Private define ------------------------------------------------------------*/
 72 #define GPIO (GPIO_USER_IO_1)
 73
 74 /* Private macro -------------------------------------------------------------*/
 75 /* Private variables ---------------------------------------------------------*/
 76 /* Private function prototypes -----------------------------------------------*/
 77 /* Public variables ----------------------------------------------------------*/
 78 /* Private functions ---------------------------------------------------------*/
 79
 80 /*-----------------------------------------------------------------------------*/
 81 static void crash()
 82 /*-----------------------------------------------------------------------------*/
 83 {
 84   volatile uint32_t *puiAddress = 0xDEADFACE;
 85   uint32_t uiData;
 86
 87   uiData = *puiAddress;
 88 }
 89
 90 /*-----------------------------------------------------------------------------*/
 91 static void ToggleGPIOProcess(void *pvArg)
 92 /*-----------------------------------------------------------------------------*/
 93 {
 94   GPIOToggle(GPIO);
 95 }
 96
```

- Find callee and caller with Linker Register (R14)
  - Callee ⇒ 0x080049d5 – 2 = 0x80049d3

```
[bellac@lappc-f533] /tmp/workarea/KIWI/cmt> arm-none-eabi-addr2line -e ../arm-gcc47-dbg/bmc_IOIF.elf 080049d3
/tmp/workarea/KIWI/cmt/../src/toggle_gpio.c:116

 90 /*---------------------------------------------------------------------------*/
 91 static void ToggleGPIOProcess(void *pvArg)
 92 /*---------------------------------------------------------------------------*/
 93 {
 94   GPIOToggle(GPIO);
 95 }
 96
 97 /* Public functions ---------------------------------------------------------*/
 98 /*---------------------------------------------------------------------------*/
 99 bool ToggleGPIOInit(void *pvArg)
100 /*---------------------------------------------------------------------------*/
101 {
102   if (!GPIOOpen(GPIO)) {
103     printf("ERROR> [Toggle-GPIO] Could not open GPIO.\n");
104     return false;
105   }
106
107   if (!GPIOSetConfig(GPIO, GPIO_CFG_MODE, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE)) {
108     printf("ERROR> [Toggle-GPIO] Could not configure GPIO.\n");
109     return false;
110   }
111
112   GPIOSet(GPIO);
113
114   printf("INFO> [Toggle-GPIO] Properly configured GPIO_USER_IO_1.\n");
115
116   crash();
117
118   return true;
119 }
```

- What you will do?
  - Create project area
  - Hello world (blinky)
  - Configuring your project
  - Edit FRU information and SDR
  - Register your sensors
  - Build and Upload IPMC firmware
  - Debugging a Bus Fault
  - Writing sensor driver

THANK YOU FOR YOUR ATTENTION