HYDROGEN SYSTEM

Time box 5 – 17-05-2011

This report covers Time box 5 in the realization phase of the Hydrogen subproject, which is a part of the overall Energy Hub project.

The project is a mandatory part of the 4th semester at the Electronic Design Engineer education at AU-IBT.

The Project has been supervised by Klaus Kolle and Morten Jakobsen both teachers at the Electronic Design Engineering program.

Lasse Lykkegaard

Dennis Thomsen

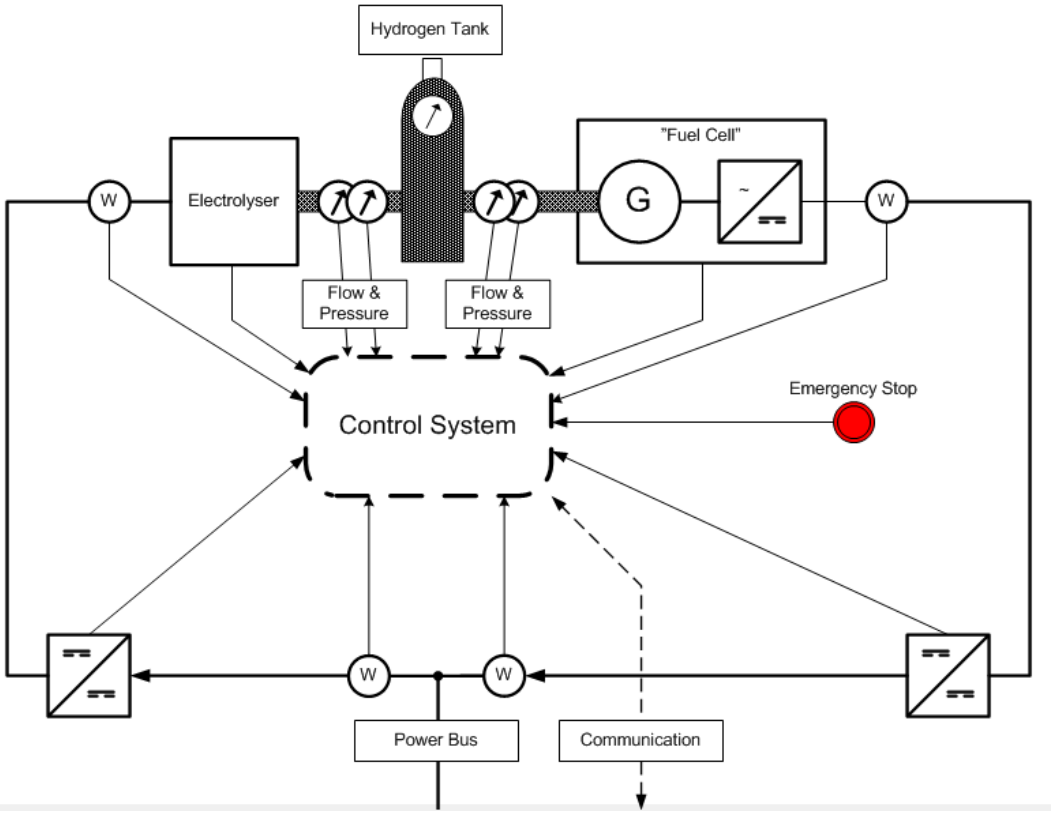
Knud Baastrup

CONTENT

System Overview

Lasse Lykkegaard

Below figure show the System Overview created during the PRO3 Launch project



Change log

This section describes some project adaptations done in PRO4 compared to PRO3.

The altering specifications and the work in the realization phase have revealed some none coherent specifications which have been altered.

The functions getSensorData() does not need the ProductionID argument supplied, when working in the data model...sensors are no longer connected to a Production system. Instead it has been supplied with a &data, where the data is written. This makes us able to track errors.
getSensorData(int SensorId, int &data)

The interface between the user-space application and the Data model was previously separated with two interfaces, one for the sensor class and one for the production system class. These two have been merged together into an h2SystemInterface class.

The State Machine diagram of the Fuel Cell and Electrolyzer should be altered to include a super state as emergency shutdown. At least the emergency shutdown should be made a blocking condition. Super state is preferred.

DEPLOYMENT PLAN

Knud Baastrup

Table 1 shows the deployment plans that were initially defined during the strategy planning of time box 1. The plan has now been updated with more details on the later product versions that in the initial plan just included a heading for the expected deliverables.

Table 1 Deployment Plan

Product Version	Functionality	Deployment Date
1	Start/Stop Electrolyzer and implement Database: The Electrolyzer can be started and stopped from a software application running in user-space on a LPC2478 Development kit. The power bus is simulated by using the XPR 60-100 power supply. Database design is settled and implemented on lene-lasse.dk Participatory session with low fidelity prototype completed. The product version consists of the following artifacts: • Electrolyzer (final version) • Converter/relay-switch (prototype 1) • LPC2478 Development kit (final version) • Driver for converter/relay-switch (final version) • H2 application (version 1) • Database design (final version) • Low fidelity prototype • Report documenting the design	15-03-2011
2	Measure efficiency of the Electrolyzer and implement dynamic web: It is possible to measure the efficiency of the electrolyzer by measuring the energy consumed versus energy produced by the electrolyzer. This requires the ability to measure voltage, current, flow and pressure. The measured data can be read-out from the H2 application sw. Dynamic web site can show measured data (data are simulated using scripts). Video presentation to cover the participatory session with low fidelity prototype. The product version consists of the following artifacts: • Current sensor (prototype 1) • Voltage sensor (prototype 1) • Flow sensor (prototype 1) • Driver for A/D converter (final version) • H2 application (version 2) • Dynamic web site (version 1) • Video presentation (final version) • Report documenting the design	29-03-2011

3	Gain CAN knowledge and finalize PHP Websites: • Establish CAN communication between 2 LPC2478 boards • Login support for PHP website (with java script validation) • Support for Mail notification for sensor data • Improve SQL query for power calculation and improve graph presentation	12-04-2011
4	Final testing of sensor hardware and get started with FPGA support • FPGA Module design including 8 channel ADC • Final testing of sensor HW • Preparation tasks for EMC test • Update of sensor data to MySQL/web server • Preparation of usability test • Read-out of efficiency data via web • Continuously update of sensor data and average filter	03-05-2011
5	CAN support, EMC verification, usability test and finalize and FPGA support: • H2 Application support for CAN bus (handler functions) • Driver for CAN Controller • EMC verification in terms of Emission and Immunity test • Usability test of dynamic H2 website • Extended test bench for FPGA based ADC • Supporting HW in terms of R-2R ladder	17-05-2011

Strategy Planning

Knud Baastrup

Timebox 5 is the last planned deployment in the H2 project and it includes therefore all the remaining requirements. The requirements ready for sign off will be listed in a Product Accept Declaration for timebox 5.

Time box 5 Specification

Knud Baastrup

This time box covers the below requirements, which must be realized in order to meet the functionality specified for product version 5 in the deployment plan in Table 1.

ID	General Requirements	Time box	Signed
G1.3	The H2 subsystem shall be physical connected to a power-bus with a nominal voltage of 48 VDC. The voltage should be within 48 ±2 VDC with a maximum superimposed AC voltage of 2 V_{pp} in accordance with the Bus Voltage Specification. The power-bus will both deliver the power to be stored as hydrogen and consume the power produced using hydrogen fuel cell.	5	
G1.4	The electrolyzer and fuel cell shall not consume or deliver more than 3KW or 62.5A.	5	
G1.5	The H2 subsystem shall monitor input/output power-bus voltage and input/output power-bus current and notify the Hub and other subscribed observers on changes in voltage or current level.	5	
G1.5.5	The H2 subsystem shall notify the Hub on changes in voltage and current level	5	
ID	**Electrolyzer Requirements**	**Time box**	
E1.1	It shall at any time be possible to start H2 production on request from Hub when all of the following conditions are fulfilled: 1. Defined delay between stop and start has passed. 2. Electrolyzer has not been manually stopped. 3. No power production ongoing. 4. Other (the system must be in ready state) Hub shall be notified once H2 production has started.	5	
E1.1.3	It shall be possible to manage a request from Hub to start H2 production.	5	
E1.2	It shall at any time be possible to stop H2 production on request from Hub either immediately or when defined delay between start and stop has passed. Hub shall be notified once H2 production has stopped.	5	
E1.2.3	It shall be possible to manage a request from Hub to stop H2 production.	5	
E1.3	The H2 subsystem shall be able to stop H2 production and notify Hub if one of the following conditions takes place: 1. Sufficient power is no longer available for efficient H2 production. 2. Electrolyzer has been manually stopped due to malfunction	5	

E1.3.1	The H2 subsystem shall be able to stop H2 production if sufficient power is no longer available for efficient H2 production.	5	
E1.3.2	The H2 subsystem shall be able to stop H2 production if electrolyzer has been manually stopped due to malfunction.	5	
E1.3.3	The H2 subsystem shall be able to notify Hub on the stop of H2 production.	5	
E1.4	It shall at any time be possible to manually stop H2 production immediately and notify Hub in case of emergency or for maintenance/test purpose.	5	
E1.4.1	It shall at any time be possible to manually stop H2 production immediately in case of emergency or for maintenance/test purpose.	5	
E1.4.1	The H2 subsystem shall be able to notify Hub on the stop of H2 production due to emergency or maintenance/test purpose.	5	
ID	**EMC1 Requirements**	**Time box**	
EMC1.1	The H2 system shall be tested for emission and comply to existing laws	5	
EMC1.2	The H2 system shall be tested for immunity and comply to existing laws	5	
ID	**IDE1 Requirements**	**Time box**	
IDE1.3	Usability test of web interface	5	
ID	**EMB FPGA Requirements**	**Time box**	
EMB1.1	The H2 system shall include a FPGA that communicate with LPC2478 Development kit using the Wishbone interface.	5	
EMB1.1.2	The Wishbone compliant A/D converter should support 8 channels	5	
EMB1.1.2	HW in terms of R2R ladder and comparators shall be used to support the A/D converter.	5	

Development Plan

All

Timebox 5	Week 18							Week 19							Week 20
Task	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M
Get knowledge and prepare for EMC verification						LK	LK	LK							
EMC Verification (Emission/Immunity)										A					
Get CAN driver up running in the H2 System	K									K	K	K	K	K	
Support for Serial, Greenscoore and Estimated Power in H2 System Interface	D								D						
Include comm_protocol class and copy H2 System Interface into Handler functions	D	D	D						D	D	D	D	D		
Finalize Test Bench for FPGA based A/D Converters				K	K	K									
Driver for FPGA based A/D Converters	L	L	L	L	L						L	L			
Supporting HW for FPGA based A/D (R-2R and comparators)														D	
Finalize Web report				A	A	A	A	A							
Perform and document usability test					A	A	A								
Create and finalize IDE report	K														
Functional Tests											A	A			
Create and write EMC report						LK	LK	LK							

L: Lasse Lykkegaard

K: Knud Baastrup

D: Dennis Thomsen

A: All

SW DESIGN

All

The software updates include the possibility to update sensor data into a MySQL server DB, which allow the data to be presented on the H2 Webpage.

The program is expanded with threads to maintain sensor data and an averaging filter to smoothen out flicker.

Driver for FPGA ADC

Lasse Lykkegaard

To unload the processor with A/D processing we have implemented an external A/D converter on a FPGA chipped board. This board is memory mapped as an External Memory piece controlled by the EMC of the LPC2478. It is necessary to enable a *chip select* more to use the external pins for memory locations.

The memory bus is heavily used in uClinux and therefore it is not advisable to change CS settings in the initialization of the driver. This is best done in U-boot, which sets up the other *low-level* registers upon boot.

Settings is shown in Figure 1 where WAITRD2 is the number of clocks the CPU waits to read the data pins, after the address pins has been loaded, and Chip-select is activated. Write is not used in this particularly driver, therefore all values regarding write is not tested. CFG chooses the bus width, in this example 16 bit.[1]

```
PINSEL4 |= 0x10000000;

EMC_STA_WAITWEN2  = 0x2;
EMC_STA_WAITOEN2  = 0x01;
EMC_STA_WAITRD2   = 0x06;
EMC_STA_WAITPAGE2 = 0x1f;
EMC_STA_WAITWR2   = 0x1f;
EMC_STA_WAITTURN2 = 0xf;
EMC_STA_CFG2 = 0x00000081;
```

Figure 1 – EMC coefficients

```
/** Adresses of the registers */
#define FPGA_BASE_ADDR  0x82000000
#define ADC_REG0       (*(volatile unsigned short *)(FPGA_BASE_ADDR + 0x000))
#define ADC_REG1       (*(volatile unsigned short *)(FPGA_BASE_ADDR + 0x002))
#define ADC_REG2       (*(volatile unsigned short *)(FPGA_BASE_ADDR + 0x004))
#define ADC_REG3       (*(volatile unsigned short *)(FPGA_BASE_ADDR + 0x006))
#define ADC_REG4       (*(volatile unsigned short *)(FPGA_BASE_ADDR + 0x008))
#define ADC_REG5       (*(volatile unsigned short *)(FPGA_BASE_ADDR + 0x00A))
#define ADC_REG6       (*(volatile unsigned short *)(FPGA_BASE_ADDR + 0x00C))
#define ADC_REG7       (*(volatile unsigned short *)(FPGA_BASE_ADDR + 0x00E))
```

Figure 2 – Registers

When defined and typecasted like in Figure 2 with the * in front, ADC_REGX acts like a variable, a hardwired pointer.

[1] See LPC2478 manual for further details.

Int x = ADC_REG1; is valid, but when dealing with functions, like copy_to_user(), that requires a pointer, this implementation makes it more difficult.

It has not been easy – especially the reset pin of the FPGA is apparently very sensitive.

The FPGA is implemented internally with a Wishbone[2] interface.

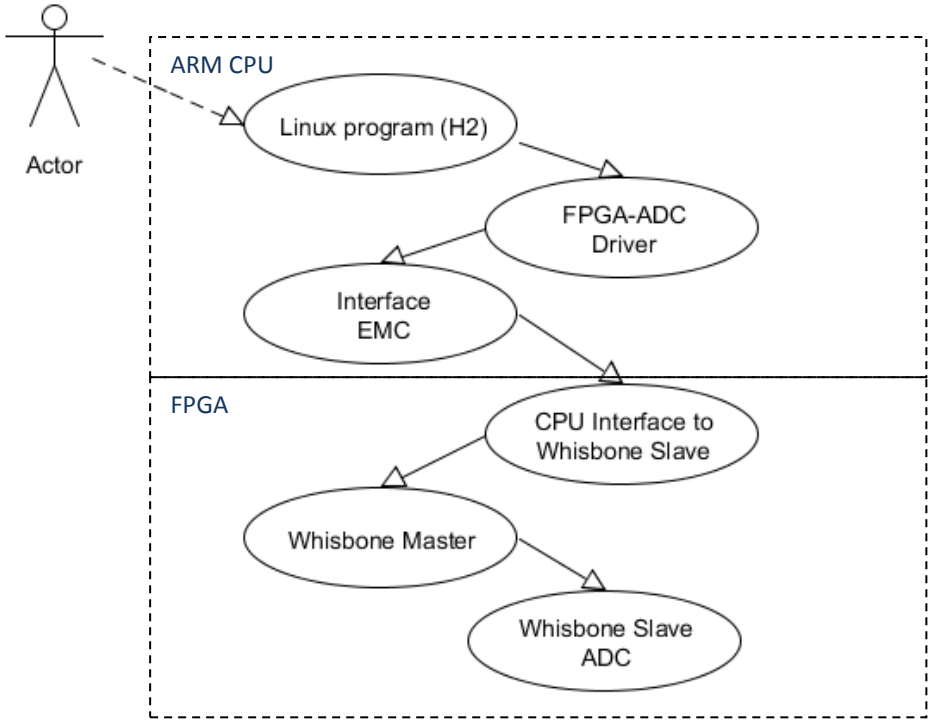


Figure 3 – a diagram of how the data is located from the external ADC

The driver is built upon the driver for the converter. Unlike the converter the FPGA-ADC driver does not have any support for a write-function. It is to come when more time is acquired – drivers should be generic and without policy.

Not Stable

The same hardware from the same bulk behaves different. Even though we test the same linux-image on different boards, the system behaves differently, and is not stable for example:

Figure 4 – Nand chip missing

[2] Whisbone is an open system bus/system-on-chip to interconnect different architectures with a common interface

The NAND Flash is not registered. Nand chip is not used in the hydrogen project – therefore this error seams uninteresting. The source of the problem could be hardware or software related. Random errors are hard to track down. Like in Figure 5 a Kernel lockup is detected when booting into uClinux– but to track down the error would be very time consuming for us.

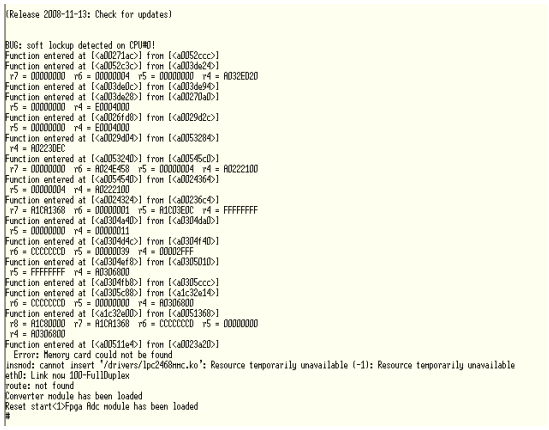


Figure 5

When this kernel lookup is detected the board will freeze unexpected at random operations. HW reset is the only solution.

Driver for CAN Controller

Knud Baastrup

The communication towards the Hub is managed via the CAN protocol, which is directly accessed and addressed from the application software even though it only manages layer 2 of the OSI model. This seems however to be common practice when working with CAN.

All teams of the Energy Hub project require a CAN driver for Linux and Team User Interface has taken the lead in providing a suitable driver for the Energy Hub project. Team User Interface have identified a driver written by a guy call Dmitry Kobylin. The driver works, but there is no documentation and the amount of code comments are very limited.

We have done some code investigation in order to figure out how it gets the job done and to get some knowledge about how the driver can be used from a user application point of view. The observations are documented below and can be used for reference when needed.

Module Initialization

The CAN driver is initialized when loaded with insmod. The CAN driver initialize two CAN devices that each is allocated a char device structure (**cdev**) and a can device structure (**candev_t**). The char device structure (**cdev**) is as usual used to register the supported file operations that for the CAN driver include the following:

```
static struct file_operations candev_fops = {
    .owner   = THIS_MODULE,
    .open    = candev_open,
    .read    = candev_read,
    .write   = candev_write,
    .ioctl   = candev_ioctl,
    .release = candev_release,
    .poll    = candev_poll,
};
```

The can device structure (**candev_t**) includes, as shown below, a 1024 bytes FIFO and as well a structure for a work queue (**read_wq**) and some wait queues (**fifoque**, **rque** and **wque**). The work queue and wait queues are required in order to service multiple processes accessing the driver simultaneously.

```
struct candev_t {
    int dev; /* Device number */
    atomic_t can_available;
    struct work_struct read_wq; /* Work queue for reading buffers*/
    wait_queue_head_t rque, fifoque;
    wait_queue_head_t wque;
    int srr_bit; /* Self reception request */
    int rfinish ;

    struct can_fifo fifo;
};
```

Finally the CAN driver is also registering an interrupt routine (**lpc2xxx_can_interrupt**) that will be scheduled each time an interrupt are generated by the CAN controller. The interrupt service routine will defer some of its work to the work queue that during initialization were registered with the function pointer **wq_read_feed**, i.e. the interrupt service routine will ensure that the function **wq_read_feed** is scheduled to complete the job.

CAN Controller Initialization

The CAN Controller is initialized when a CAN device (/dev/can0 or /dev/can1) is opened from a user space application. The following behavior is configured in the relevant registers.

Bitrate (CANxBTR)

The bitrate is calculated and configured as:

PCLK_CANx / (BRP x (1 + TESG1 + TESG2)) = 48MHz / (14 x (1 + 11 + 2)) = 245 kbit/s

Interrupts (CANxIER)

Interrupts are enabled so an interrupt is generated each time the receive buffer is full and each time a message has been sent from one of the 3 transmit buffers.

Read and Write

Read is performed via the file operation **candev_read**. The user space application can specify the number of messages to be read and **candev_read** will copy the number of messages to userspace or at least the amount of messages available in the FIFO. If the FIFO is empty, the process attempting to read will be put to sleep and awakened by the work queue **wq_read_feed** once some data is available.

Write is performed via the file operation **candev_write**. The user space application can specify one message to be written and **candev_write** will copy the message to kernel space and transmit the message via one of three transmit buffers. If all the transmit buffers are full, the process attempting to write will be put to sleep and awakened by the interrupt service routine (**lpc2xxx_can_interrupt**) once there is room in one of the transmit buffers.

Interrupt Service and Work Queue

The interrupt routine is scheduled each time a message has been either transmitted or received to/ from the CAN Controller.

If a message has been transmitted a **wake_up_interruptable** macro is called in order to wake up the processes in the **wque** that has been put to sleep because the transmit buffers were full.

If a message has been received, the message is taken out of the receive buffer and added to the FIFO (**can_fifo**) that was allocated as part of the can device structure (**candev_t**). Before the interrupt service routine completes, the work queue (**wq_read_feed**) will be scheduled (using the macro **schedule_work**) to complete the task.

The task of the work queue (**wq_read_feed**) is mainly to wake up the processes in the **rque** with the **wake_up_interruptable** macro. The work queue will then feed the waiting processes with the data stored in the FIFO and when done it will simply go to sleep using the **wait_event_interruptible** macro and await that it is scheduled once again from the interrupt service routine, i.e. when new data are present in the FIFO.

Acceptance Filter

By default, the driver is initialized to use the Acceptance filter in bypass mode, which accepts all messages on the CAN bus and as well allow the raw message ID to be indexed in an ID look-up table.

IOCTL Support

The driver supports some IOCTL commands, which includes the following commands:

CANLPC2000_IOCRESET:	Reset CAN Controller
CANLPC2000_IOCSBITRATE:	Set bitrate (CANxBTR)
CANLPC2000_IOCSMODE:	Set mode (CANxMOD)
CANLPC2000_IOCSSRR:	Set Self Reception Request (transmitted message will be copied to receive buffer)
CANLPC2000_IOCSAFTABLE:	Configure the Acceptance Filter. (allow the Acceptance filter to operate in FullCAN Mode)

Comm. Protocol Handler Functions

Dennis Thomsen

Interacting with the Hub, and interface devices is done using the comm. Protocol, written by the Hub team with the following functions:

```
sendAllDevicesStop();
sendSerialNumberToEnergyHub();
sendAddressFromEnergyHub(char serialNumber[SERIAL_NO_LENGTH], char
address);
```

```
sendSystemMsgFromEnergyHub(int targetId, int data, sys_msg_from_eh_t
command, DeviceManager *devMan);
sendSystemMsgToEnergyHub(status_t status, char _data[4]);
sendExpPwrMeasurement(float expPwr);
sendPwrMeasurement(float voltage, float current);
sendMsgFromUserInterface(int targetId, msg_from_ui_t command);
sendMsgToUserInterface(int data, msg_to_ui_t command, char
_data[7]);
sendDevSpecMeasurement(int devSpecMeasId, char data[DATA_SIZE], int
dataLength);
```

For those functions in the protocol that send data, to our device, functions need to be written that handle that incoming data, and act according, these function are referred in the future as handler functions, the following functions from the comm. Protocol needs to have a handler function written for it

```
sendAllDevicesStop();
sendAddressFromEnergyHub(char serialNumber[SERIAL_NO_LENGTH], char
address);
sendSystemMsgFromEnergyHub(int targetId, int data, sys_msg_from_eh_t
command, DeviceManager *devMan);
sendMsgFromUserInterface(int targetId, msg_from_ui_t command);
```

What the handler functions should do is mostly pretty obvious from the names,

The handler function for `sendSystemMsgFromEnergyHub` needs to act on the command sent from the hub, there are three commands it can send `STOP, START, RECONNECT,` upon

In the Communication Protocol document made by the Hub team, it specified how code is to act: we need to continually run the `sendSerialNumberToEnergyHub` function until the hub responds with the address upon which time we need to send the systems Greenscore with `sendSystemMsgToEnergyHub` before 500 milliseconds elapse, after which either `sendExpPwrMeasurement` or the `sendPwrMeasurement` message should be send within 500 milliseconds or the device is disconnected from the hub, additional the system is to keep sending one of those messages every 100 milliseconds , depending on whether the system is actively producing power at the moment, in order to achieve this a thread is to be used that will sleep for 100 milliseconds after it has run though the code, and as the system isn't disconnect until 500 milliseconds pass it don't matter if it gets a bit delayed.

Below in Figure 6 the current version of the handler function for the `sendSystemMsgFromEnergyHub` can be seen

```
int sysMsgFromEH_handler(msg_t msg) {
    /* Variables */
    int result;
    int baseAddr = SYS_MSG_FROM_ENERGY_HUB;
    sys_msg_from_eh_t status;

    /* Extract status */
    status = (sys_msg_from_eh_t)msg.data[msg.dataLength - 1];

    /* Check valid */
    if ((status < 0) || (status > LAST_SYS_MSG_FROM_EH - 1)) return -1;

    switch (status) {
    case STOP:
        result = h2If.stopProduction(ELECTROLYZER_ID);
        if (result == SUCCESS)
                {
                    //send msg to hub
                    commProto->sendSystemMsgToEnergyHub(STOPPED, NULL)
                }
        break;
    case START:
        result = h2If.startProduction(ELECTROLYZER_ID);
        if (result == SUCCESS)
                {
                    //send msg to hub
                    commProto->sendSystemMsgToEnergyHub(CONNECTED, NULL)
                }
        break;

    default:
        return -1;
    }
    /* Return success */
    return 0;
}
```

Figure 6

There is not yet a response to the reconnect command as it is a recent addition to the hubs commands

A `installFunction` from the comm. Protocol is used to ensure the correct handle function is called for the message received.

```
void H2Handler::SendSensorData()
{
     int voltage = 0;
     int current = 0;
     int flow = 0;
     int expPwr = 0;

   while(1)
   {
        if (electrolyzer->PB_converter->getConverterStatus()) {
            h2If.getSensorData(CURRENT_OUT_SENSOR_ID, current);
            h2If.getSensorData(VOLTAGE_OUT_SENSOR_ID, voltage);
            h2If.getSensorData(FLOW_OUT_SENSOR_ID, flow);

            //convert int flow to char data
            char buffer [8];
            sprintf (buffer, "%X", flow);

            sendPwrMeasurement(voltage,current);

            // devSpecMeasId = 0,
            sendDevSpecMeasurement(0, buffer, 4);
        }else{
            sendExpPwrMeasurement(ExpectedPower);
        }
   // wait for 100 milli sec
   usleep(0100000);
    }
}
```

Figure 7: code for continually sending sensor data to hub

Figure 7 shows the code that will be running in its own tread constantly sending data to the hub, it should be noted that as of now the code is not complete or working, therefore changes' will most likely happen as we attempt to get a working connecting with the other teams.

ADC Driver update

Dennis Thomsen

A few minor changes have been made to the adc driver for the on board adc's, pulldown mode have been selected for the input rather than the default pullup, this was done much the same way the pins where selected to be adc's inputs.

```
// enable pulldown
if ( channel <= 3){
    m_reg_bfs(PINMODE1, pulldown_bits[channel]);

    } else if ((channel == 4) || (channel == 5)){
    m_reg_bfs(PINMODE3, pulldown_bits[channel]);

    } else if ((channel == 6) || (channel == 7)){
    m_reg_bfs(PINMODE0, pulldown_bits[channel]);
    }
```

```
static u32 pulldown_bits[] =
{
  (11 << 14), // ad0.0   P0[23]
  (11 << 16), // ad0.1   P0[24]
  (11 << 18), // ad0.2   P0[25]
  (11 << 20), // ad0.3   P0[26]
  (11 << 28),   // , ad0.4 P1[30]
  (11 << 30),   // , ad0.5 P1[31]
  (11 << 24),   // , ad0.6 P0[12]
  (11 << 26),   // , ad0.7 P0[13]
};
```

Figure 8: ADC code to enable pulldown resistors on inputs channels

This change was made because there was a problem with the input signals when connected to the input pins, the signals could not go under 175mV, but after the change they could go as far down, as 30mV.

HW DESIGN

FPGA Test Bench for 8-ch ADC

Knud Baastrup

It is recommended to re-cap the FPGA module design included in the timebox 4 deployment document before further reading.

In Timebox 4 we included the complete FPGA module design and managed as well to complete the VHDL coding and perform some overall verification in a Test Bench. The FPGA solution were as well deployed on the deployment meeting by using the R-2R ladder and the two comparator channels on the Spartan 3A Extension board.

In this timebox 5, we have extended the Test Bench by including a more accurate LPC2478 CPU timing that should allow us to find the optimal values for the EmcStaticWaitRd2 register. We have furthermore added a test specific for the ADC that includes some of the core functionality provided by the 8-ch ADC.

LPC2478 CPU Timing

The following timing was included in the Test Bench to get a more realistic simulation that could allow us to determine for how long time we need OE (Output Enable) to stay low, i.e. to find the value to be configured in the EmcStaticWaitRd2 register.

```
package arm_emc_package is
    --! cycle @ 57.6 MHz
    constant CYCLE      : time := 17.36 ns;

    --! EMC timing, adjust to fit electricaldatasheet
    --! Read timing
    constant WAIT_RD        : integer := 5;
    constant WAIT_OEN       : integer := 1;
    constant Tr_CSLAV       : time := 2.54 ns; -- Max
    constant Tr_CSLOEL      : time := (0.49 ns + (WAIT_OEN * CYCLE)); -- Max
    constant Tr_OELOEH      : time := (-0.59 ns + (WAIT_RD - WAIT_OEN + 1) * CYCLE); -- Min
    constant Tr_AM          : time := ((WAIT_RD - WAIT_OEN + 1) * CYCLE - 12.70 ns); -- Max
    constant Tr_OEHANV      : time := 0.20 ns; -- Typ
    constant Tr_CSHOEH      : time := 0 ns; -- Typ
```

Figure 9 shows the read-out of the 8 ADC registers maintained by the WB_ADC_8ch_Slave entity. It is the same read-out as provided in timebox 4, but now with accurate CPU timing. The ADC registers are initialized with a number corresponding to the number of the register.

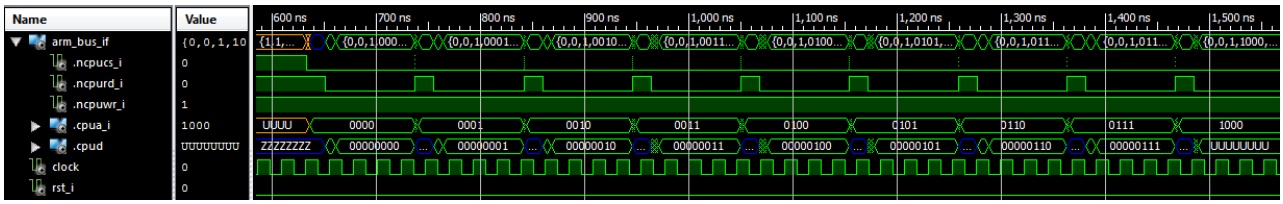


Figure 9: Read-out of the 8 ADC registers via Wishbone Interface using accurate CPU timing

Figure 10 shows a zoom-in on Figure 9 where we can see that valid data is present on the data bus after approximately 26 ns. This should (theoretically) allow the EmcStaticWaitRd2 register to be configured to wait just two CPU clock cycles because the CPU clock period is around 17 ns (1 / 57.6 MHz).

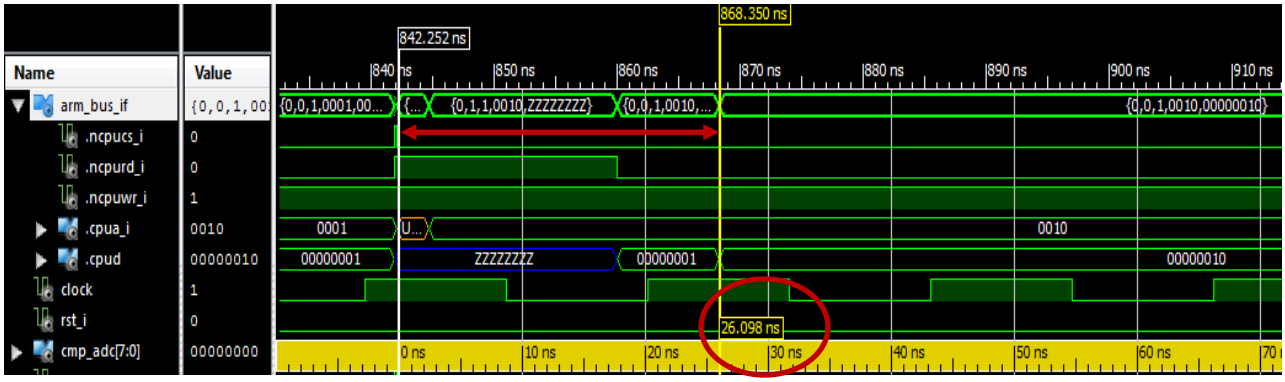


Figure 10: Zoom in to show that data is ready 9 ns after OE is asserted

Real test have shown that EmcStaticWaitRd2 must be configured for around 4 to 5 clock cycles to prevent read-out of wrong data. This is due to the longer clock period on Spartan 3A compared to LPC2478 that can result in up to two additional clock cycles.

Test Bench for ADC

Fejl! Henvisningskilde ikke fundet. shows a snapshot from the simulation performed using the Test Bench. We can follow the 8 step building up the r2r bit pattern and also see how the adc_bin is assigned the final value when the done_tick goes high.

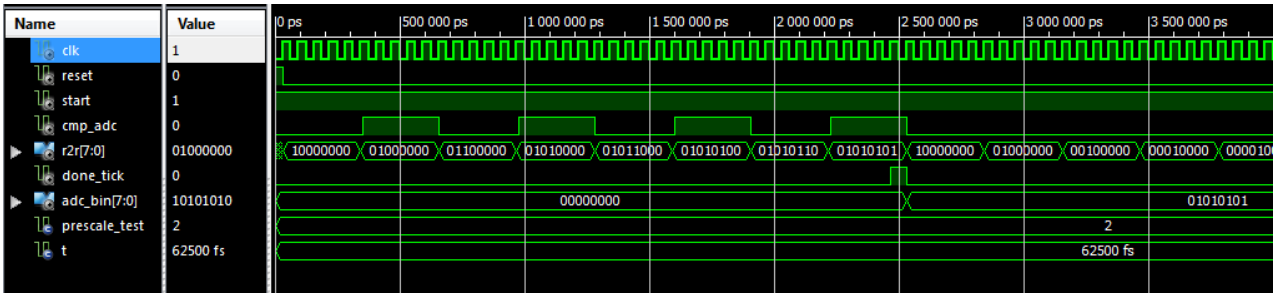


Figure 11: Snapshot of the simulation performed using the Test Bench for the ADC module

Supporting Hardware for R-2R ladder

Dennis Thomsen

As said under FPGA Test Bench for 8-ch ADC, we made use of a board with an r-2r ladder and two comparators. Although to fully implement all the sensors via the Spartan 3A board, there should at least be 7 comparators, but in the interest of saving time, we opted instead to just an old Spartan 3A Extension board, that was made previously, to simply give a proof of the concept.

Such a ladder have a 2^n-1 resolution (n = number of bits in the counter), with 8 bit this means a resolution of 255, given that each step corresponds to 0,013V.

EMC

In time box 4, we managed to get a functional approval of the H2 control system that currently can manage an electrolyzer. The H2 control system has been prepared for the management of a "fuel cell" as well and the design allows a 100% reuse of the HW already prepared for the electrolyzer. Before replicating the HW for the "fuel cell", it makes good sense to check to what extend we comply towards the EMC directive.

Test Requirements

Knud Baastrup

The H2 control system can be classified as an apparatus, which is a single functional unit intended for an end-user. Apparatus must be CE marked and comply with the EMC directive 2004/108/EC.

The list of harmonized standards covered by the EMC directive 2004/108/EC is available via the official website of the European Commission and can be located via below direct link:

http://ec.europa.eu/enterprise/policies/european-standards/documents/harmonised-standards-legislation/list-references/electromagnetic-compatibility/index_en.htm

The harmonized standards are the standards that have been published in the Official Journal of the European Union and cover a huge amount of product specific standards and as well some generic standards.

Browsing and searching through the product specific standards did not reveal any standard specific for a H2 control system, so we will go ahead and for now use the generic standards.

The generic standards are split into standards for "residential, commercial and light-industrial environments" and standards for just "industrial environments". The H2 control system falls into the category of "residential, commercial and light-industrial environments" as it will be located at AU-HIH and maybe in the future in some residential or commercial environments.

The generic standards for "residential, commercial and light-industrial environments" include:

- EN 61000-6-1: Immunity
- EN 61000-6-3: Emission

It will be a huge task and beyond the scope of this project to complete a full-blown EMC test and get to the point where we can claim the CE mark for the H2 control system. We will therefore in this project limit the effort to some indicative tests for immunity and radiated emission.

Electromagnetic immunity

The test requirements for immunity covers immunity towards the cabinet it selves, signal ports, DC input/output ports and AC input/output ports. The test requirements are defined in a number of basis standards that have been referenced in the generic standard for immunity.

The Hydrogen system consists of an enclosure port (the cabinet it selves), signal ports (CAN Bus and Ethernet), DC input ports (9 VDC supply and 48 VDC from Power Bus).

We will however only perform a very limited amount of immunity tests, which include the immunity towards the enclosure port and only the immunity towards an AM modulated high frequent electromagnetic field well knowing that a complete immunity test also should include the immunity towards a low frequent magnetic field and electrostatic discharge.

The immunity requirements for enclosure port have been listed in Table 1.

	Fænomen	Testkrav	Basisstandard	Bemærk-ninger	Fejl-kriterium
1.1	LF magnetfelt	50 Hz 30 A/m	EN 61000-4-8	Se note 1	A
1.2	Radiofrekvent elektromagnetisk felt AM moduleret	80 - 1000 MHz: 10 V/m 1.4 - 2.0 GHz: 3 V/m 2.0 - 2.7 GHz: 1 V/m 80% AM (1kHz)	EN 61000-4-3	Se note 2	A
1.3	Elektrostatisk udladning	±4 kV kontaktudladning ±8 kV luftudladning	EN 61000-4-2		B

Note 1. Kun for udstyr som indeholder magnetisk følsomme komponenter (Hall element, elektrodynamisk mikrofon o.l.). Der er specielle regler for katodestrålebilledrør, CRT.
Note 2. I de frekvensbånd, der benyttes til FM radio og TV udsendelse, er kravet reduceret til 3 V/m.

Table 1: Immunity requirements for enclosure ports

The immunity for an electromagnetic field should, according to the standard, be tested for the frequency range from 80 MHz to 2.7 GHz and the electromagnetic field should be 80% modulated with 1 kHz sinus. The H2 control system should withstand an electromagnetic field according to Table 1 and the frequency sweep should be performed with a maximum step size of 1%.

Official tests are performed in a radio dead chamber where the test object is exposed from an antenna, but indicative tests can be performed using a TEM cell.

We will use a TEM cell located at HIH and apply the electromagnetic field using a HF signal generator together with an amplifier.

The H2 control system must be supervised during the exposure and this will be achieved by performing some continuously read-outs of the sensor data using the H2 application.

Radiated emission

The test requirements for emissions covers emission towards the cabinet it selves, signal ports, DC input/output ports and AC input/output ports. The test requirements are defined in a number of basis standards that have been referenced in the generic standard for emission.

The Hydrogen system consists of an enclosure port (the cabinet it selves), signal ports (CAN Bus and Ethernet), DC input ports (9 VDC supply and 48 VDC from Power Bus).

We will, as in the case for the immunity tests, only perform a very limited amount of emission tests, which include the emission from the enclosure port.

The emission requirements for enclosure port have been listed in Table 2.

Fænomen	Frekvensområde	Grænseværdier	Basis-standard	Bemærk-ninger
Radio-frekvente elektro-magnetiske felter	30 - 230 MHz 230 - 1000 MHz	30 dB(µV/m) i 10 m afstand 37 dB(µV/m) i 10 m afstand	CISPR 16-2-3	Statistisk evaluering af måle-resultater for serie-produceret udstyr
	1 - 3 GHz	70 dB(µV/m) peak 50 dB(µV/m) average i 3 m afstand	CISPR 16-1-1 CISPR 16-1-4 CISPR 16-2-3	
	3 - 6 GHz	74 dB(µV/m) peak 54 dB(µV/m) average i 3 m afstand		

Table 2: Emission requirements for enclosure ports

The measurement of radiated emission should according to the standard cover the frequency range from 30 MHz to 6 GHz. The H2 control system must stay below the field stated in Table 2.

Official measurements are performed in a 10-Meter Chamber, but indicative measurements can be performed in a TEM cell. The TEM cell can in this case be used to determine the critical frequencies, which afterwards can be re-checked in an official 10-Meter Chamber.

We will perform the indicative measurements for the H2 control system by using the TEM cell located at AU-HIH. This TEM cell should be able to measure frequencies up to ~200 MHz as defined by its physical dimension. The critical frequencies are identified using a bandwidth amplifier and a spectrum analyzer.

TEST CONFIGURATION

Lasse Lykkegaard

In the Hydrogen System multiple modules exist, Control module, 2 x Gain and Level modules, Voltage and Current sensor module, Relay module, flow sensor module. These modules combined are our Hydrogen system. The Device Under Test is our hydrogen system.

The hydrogen system is rigged up, as it will be in a regular situation. Except for the Energy bus and the CAN-connection to the System Hub, neither is yet fully operational. The device is supplied from a USB and from an external 9VDC supply, both error-prone when considering emission.

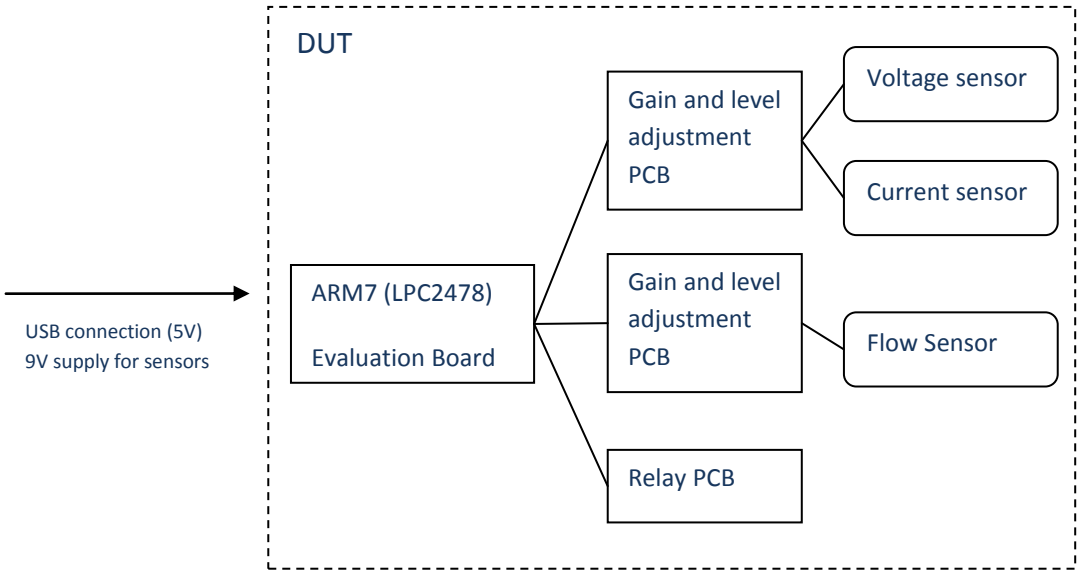


Figure 12 – Device Under Test (DUT)

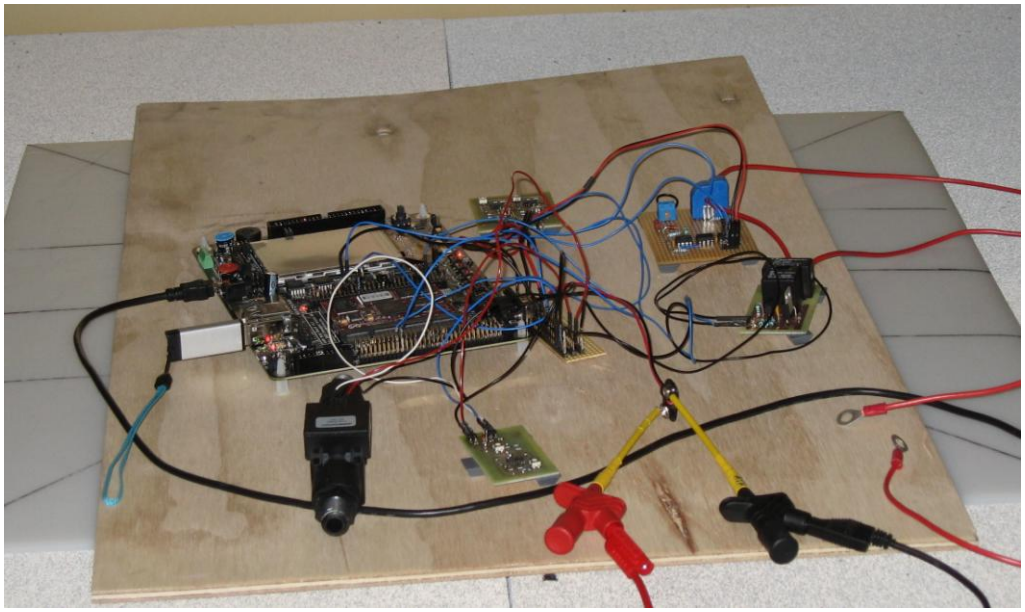


Figure 13 - DUT

TEST RESULTS FOR EMISSION

Lasse Lykkegaard

The TEM-cell is connected to an analog spectrum analyzer with a bandwidth of 80-1000MHz required according to the standard. A regular oscilloscope is not used due to the high sampling frequency needed to analyse 1 GHz (2 GHz sampling). The racking generator is not used.

The frequency spectrum is scanned for any abnormal frequencies present in the empty TEM-cell. If such a frequency is present, the setup must have an error or an antenna-wire carries a signal into the cell.

The hydrogen system is placed in the cell, with external wires (as short as possible) then the spectrum is swept unpowered. The external wires act as antennas carrying noise from the outside into the cell.

No frequencies are higher than the allowed according to 61000-6-3. The local radio stations are present in the sweep around 100Mhz, it's clearly visible that the external wires directs radio signal into the TEM-cell. In the 100-1000MHz area the GSM cellular phone network is also present around 900.

With the system turned on, several other frequency components emerged. The 48 MHz from the USB and the 72 MHz is merely visible. At 200 MHz a peak at 200 MHz and lot of harmonics with 6 MHz spacing can be seen. We cannot link this component to any distinct components of the system, other than the fact that our TEM-cells are not rated for more than 200 MHz, beyond 200 MHz reflections starts to occur in the cell. This might be the reason for us having these frequencies. Another possibility is the PLL-chip (phase locked loop) which multiplies the crystal-frequency to about 288 MHz, before dividing again.

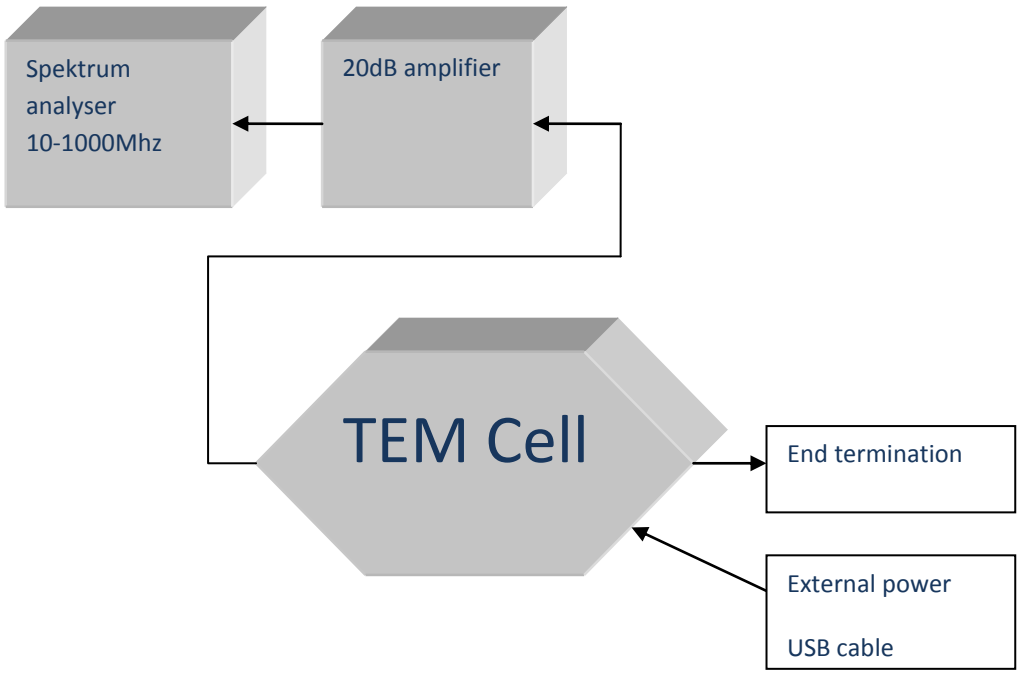


Figure 14 – Test setup Emission

Frequency [MHz]	Readings	Corrected for 20dB amp
0-1	++80dBµV	
6	28	8
7	20	0
90	36	16
91	20	0
95	15	-5
100	20	0
102	25	5
105	21	1
108	35	15

Table 3 – Unpowered Sweep

Frekvens	By	Stationsnavn
89.5	Herning	Radio Aura
90.2	Mejrup	DR P1
92.9	Mejrup	DR P3
96.2	Herning	Radio Midtjylland Classic
98.5	Mejrup	DR P4
100.3	Mejrup	DR P2
107.8	Herning	Radio M

Table 4 – Radio stations in mid-west Jutland

The spectrum analyzer seems to be need calibration – the center frequency floated a bit, and did not fit expected values exactly – but it is certain that high energy frequencies in Table 3 is radio stations.

Around zero a very large frequency is present. Probably, 50 Hz from AC-grid, the analyzer is not meant to measure in this area, neither does the standard state anything of testing with very low frequencies.

Any frequencies present in Table 3 are removed in Table 5.

Frequency [MHz]	Readings	Corrected for 20dB amp
77	30	10
150	33	13
140-246 (spaced by 6 MHz)	21	1
294	39	19
274-232 (spaced by 6 MHz)	25	15
500	30	10

Table 5 – Powered circuit

If the hydrogen system had had the possibility to switch a large current inside the TEM-cell. Burst transients would occur, but all cable conducted noise ads to the total noise emitted by the system.

We do not perform a test of conducted noise.

On Figure 15 the noise component of 72 MHz is visible. The 72 MHz is likely to come from the ARM 7 CPU which is running 72 MHz

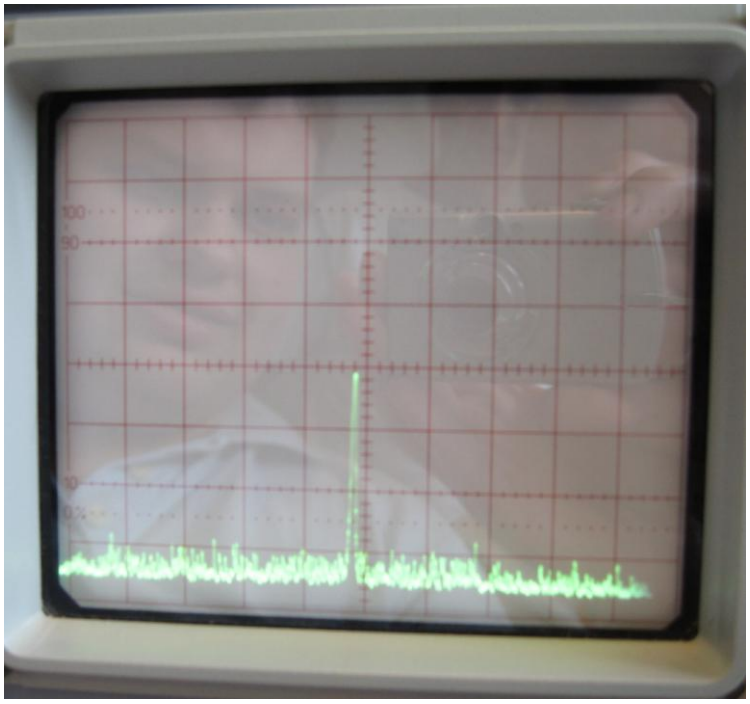


Figure 15 - Noise component of 72 MHz

CONVERSION FROM TEM-CELL TO 10M

Dennis Thomsen

Based on the voltage measured ($V_{measured}$) from the DUT in the TEM-cell, a conversion to what it would correspond to, in an official 10m test can be made.

However it should be stressed, that this is not a 100% accurate conversion, and it can therefore only be used for indicative measurements, an error margin of up to as much as 7 dB is to be expected.

Following formula is used[3]:

$$E_r = \frac{b \cdot 377}{\lambda \cdot r \cdot 50} \cdot \sqrt{\frac{2 \cdot G}{3}} \cdot V_{\text{measured}}$$

$$\Updownarrow$$

$$E_r = 20 \cdot \log\left[\frac{b \cdot 377}{\lambda \cdot r \cdot 50} \cdot \sqrt{\frac{2 \cdot G}{3}}\right] + V_{\text{measured}}$$

Where,

b: height from the inner conductor to the outer conductor in the TEM-cell. For the one we used, to the Emission test b = 0.3 m.

r: distant of the measuring site, as the standard measuring site is 10m, r = 10m.

λ: the wavelength, can be calculated with $\lambda = \frac{v}{f}$, where v is the speed of the wave and f is it frequency, f is known from the measurements and we can look up v, as we know we are dealing with electromagnetic radiation, we can use c (the speed of light, which is a electromagnetic wave) which is approximately 300,000 km/s.

G: is a gain factor. G = 1.6.

$$b := 0.3m \quad G := 1.6 \quad r := 10$$

$$E_r(f, V_m) := 20\log\left(\frac{b \cdot 377}{\frac{c}{f} \cdot r \cdot 50} \cdot \sqrt{\frac{2 \cdot G}{3}}\right) + V_m$$

$$f := 77MHz \qquad V_m := 10$$

$$E_r(f, V_m) = -14.436$$

Figure 16: Example of Conversion calculation

[3] See **Fejl! Henvisningskilde ikke fundet.**

Frequency [MHz]	$V_{measured}$ [dBµV]	E_r [dBµV/m]
77	30	-14.4
150	33	-5.6
140	21	-18.2
246	21	-13.3
294	39	6.2
274	25	1.6
232	25	0.1
500	30	1.8

Table 6: Conversion results.

As can be seen from Table 6 we get several negative results at 10m, we take this to mean that at 10 meters, these emissions would already be undetectable when looking in the µV range, even when accounting for the 7dB error margin, none of the emissions go over the half-way point of what they are allowed.

TEST RESULTS FOR IMMUNITY

Lasse Lykkegaard

Success criteria for Immunity test:

The system shall maintain operational at all time. The measured values of the A/D converter may increase up till 5 % of full range.

The large TEM-cell placed in FAB-lab is used for this test.

The regulator- and measurement tool for electric field in the TEM-cell is not functioning, therefore we adopted an alternative solution / calculation.

Test object are placed between the shield (outer wire) and the inner wire. Between these wires 1.5 V are applied. In the TEM-cell the distance between plates are 50 cm, making the electric field 3 V/M. 3 V/M is described in the standard as suitable for testing Light Industry/Household products.

When exciding the oscilloscope frequency range (60 MHz), we have no track of the electric field and have to assume that it is linear. The signal is 80% AM modulated with 1 kHz.

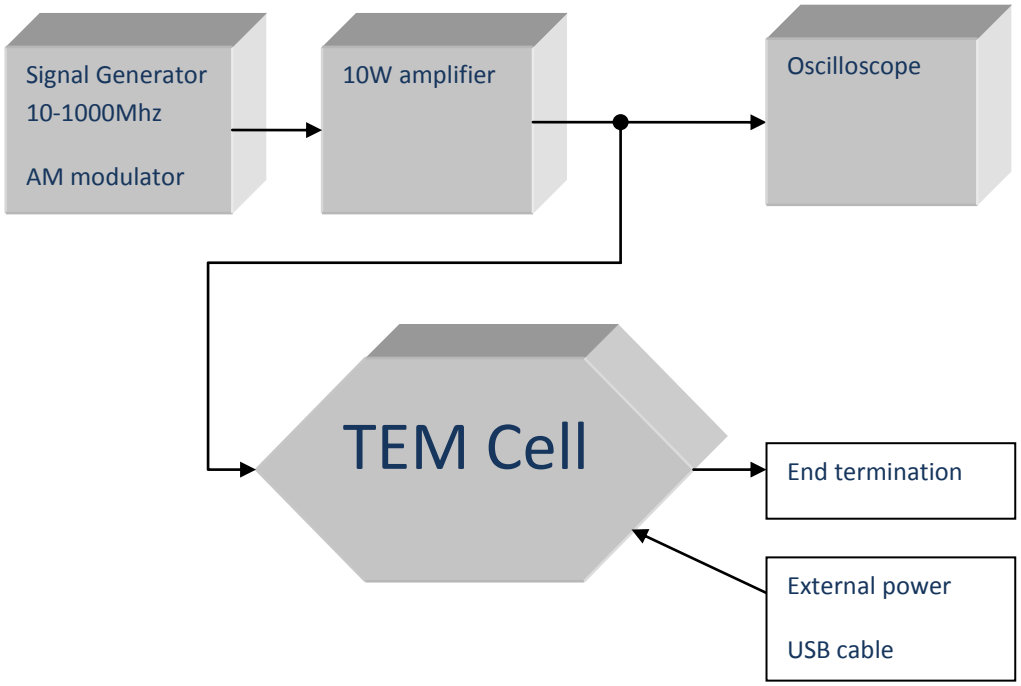


Figure 17 – Test Setup Immunity

Frequency [MHz]	Voltage Sensor [mV]	Flow Sensor [mL/m]	Current Sensor [mA]	System functionality
10	45100	28	110	Not affected
80	45110	80	124	--------"-------
150	45080	61	130	--------"-------
200	45000	46	96	--------"-------
300	45105	25	90	--------"-------
400	45150	30	112	--------"-------
500	45097	21	102	--------"-------
700	45054	39	147	--------"-------
1000	45110	79	132	--------"-------

Table 7 – Immunity test

Voltage sensor range is 45000-51000 and the Current and flow sensor is between 0 and 10000. None of the sensors alters significantly.

REFERENCES

- EUDP: http://www.eudp.net
- Dansk Radio http://www.dkradio.dk/lrringko.htm
- Word document: Udstrålingsmåling med en TEM-celle. Provided by Per Lysgaard