Communicating Process Architectures 2007
Alistair A. McEwan, Steve Schneider, Wilson Ifill, and Peter Welch
IOS Press, 2007

A Versatile Hardware-Software Platform for In-Situ Monitoring Systems

Bernhard H. C. SPUTH, Oliver FAUST, and Alastair R. ALLEN
Department of Engineering, University of Aberdeen, Aberdeen AB24 3UE, UK
{b.sputh, o.faust, a.allen}@abdn.ac.uk

Abstract. In-Situ Monitoring systems measure and relay environmental parameters. From a system design perspective such devices represent one node in a network. This paper aims to extend the networking idea from the system level towards the design level. We describe In-Situ Monitoring systems as network of components. In the proposed design these components can be implemented in either hardware or software. Therefore, we need a versatile hardware-software platform to accommodate the particular requirements of a wide range of In-Situ Monitoring systems. The ideal testing ground for such a versatile hardware-software platform are FPGAs (Field Programmable Gate Arrays) with embedded CPUs. The CPUs execute software processes which represent software components. The FPGA part can be used to implement hardware components in the form of hardware processes and it can be used to interface to other hardware components external to the processor. In effect this setup constitutes a network of communicating sequential processes within a chip. This paper presents a design flow based on the theory of CSP. The idea behind this design flow is to have a CSP model which is turned into a network of hardware and software components. With the proposed design flow we have extended the networking aspect of sensor networks towards the system design level. This allows us to treat In-Situ Measurement systems as sub-networks within a sensor network. Furthermore, the CSP based approach provides abstract models of the functionality which can be tested. This yields more reliable system designs.

Keywords. Embedded systems, System on Chip, Network on Chip, Hardware Software co-Design, Multi-core, Water Monitoring, in-situ sensors, libCSP2

Introduction

Clean drinking water is one of the most important, if not the most important food for humans and animals alike [1]. Furthermore, it is under constant danger of being polluted by environmental threats [2]. This is the reason why 9 institutions from 6 European countries formed a consortium to carry out the WARMER (WAter Risk Management in EuRope) project. The WARMER project is funded by the Sixth Framework Programme for European Research and Development (FP6). FP6 emphasises the problems of food quality, and pollution of the environment. WARMER is a follow up of SEWING (System for European Water monitorING) [3]. The focus of the SEWING consortium was the development of sensors to measure water quality. WARMER aims to enhance the work done by the SEWING consortium by creating a flexible in-situ monitoring system (IMS) and integrating remote sensing measurements obtained from satellites.

Figure 1 shows a brief overview of the system proposed by the WARMER project. The system consists of the following components:

- In-situ measurement systems (IMS) — The IMS measures the properties of the environment through its sensor probes. The obtained measurement data is interpreted

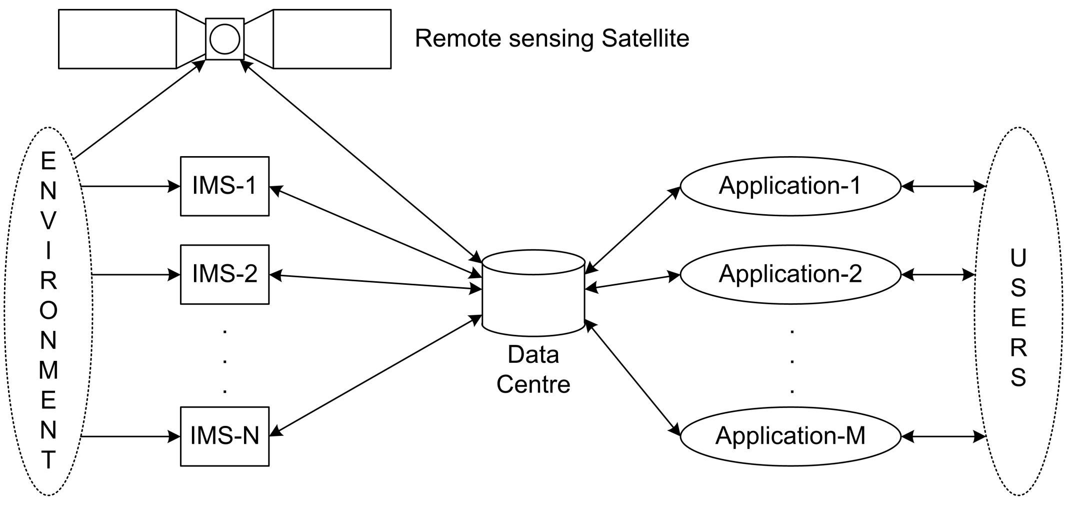

Figure 1. Overall system proposed by the WARMER project

by a built-in processing platform. The processing platform decides whether or not a critical level of pollution is reached. Under normal conditions, i.e. no critical level of pollution is detected, the measurement system sends its sensor data at predefined intervals, for instance every 30 minutes, to the Data Centre. However, if a critical level of pollution is detected, the measurement system sends a direct alarm to the Data Centre. This is only a simple example, naturally one should be able to define much more complicated monitoring schemes, for instance the measurement result of one sensor may influence the measurement strategy of another one.

- Remote sensing Satellite — It periodically acquires images of the wider area of interest. Unfortunately, these images have a low resolution, one pixel represents a square of $100\text{m} \times 100\text{m}$, and the update frequency varies between 1 and 3 days depending on the location of interest. This limits the use of remote sensing satellites. However, the WARMER consortium wants to combine satellite and IMS data to offer a more complete overview. In the following text the satellite is of no concern.
- Data Centre — A Data Centre aggregates satellite images and measurements from in-situ measurement systems over a long period of time. Furthermore, it analyses the data and exposes interfaces for user Applications. The interfaces provide access to the analysis results and the raw data. Long term data integrity is very important in order to detect slow degradations in the environment.
- Applications — These are the user interface of the system. Both IMS and Data Centre operate without human participation. The Applications interact with the Data Centre to obtain the measurement data of interest and present it to the users.

The project is interdisciplinary involving specialists in chemistry [4], environmental engineering [1,2], remote sensing [5], computer science, electronics [6], and semiconductor technology [7]. These requirements justify the large group of international collaborators.

With this paper we propose the creation of a new processing platform for the IMS. The IMSs will be deployed at remote locations, such as river beds and drinking water reservoirs. This implies that batteries power the system. The total cost of ownership for such a monitoring solution depends largely on the length of the IMS service interval. Needless to say, a longer interval is more cost effective. It is therefore of great importance that the IMS consumes as little power as possible. Before one can think of optimising the power consumption of a system, it is necessary to understand the role of the processing platform within the IMS.

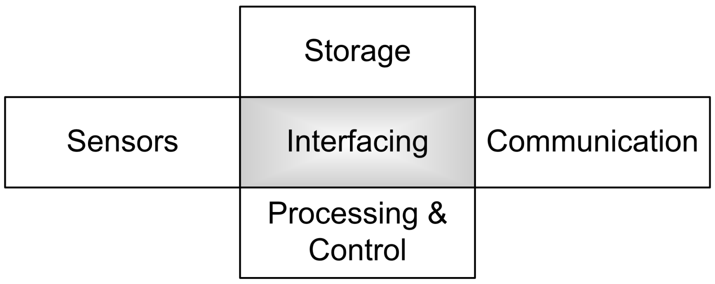

Figure 2. The building blocks of the in-situ measurement station

Figure 2 gives an overview of the IMS structure, it consists of four major building blocks: Sensors, Storage, Processing & Control, and Communication. Currently these building blocks are implemented as individual hardware components. These hardware components have incompatible and sometimes conflicting interfaces. Therefore, elaborate interfacing or glue logic is necessary to enable data exchange between these components. In general the system design has a low level of integration. This low level of integration directly translates into high energy requirements. Furthermore, there is no clear strategy which outlines the interaction between the individual components. This makes the whole design very inflexible and error prone.

Our goal is to outline a flexible design strategy which leads to energy efficient systems. In effect we propose a design strategy for NoC (Network on Chip) [8,9]. In a hardware and software co-design environment NoCs consist of networked hardware and software processes. We abstract both process types and model the network in terms of Communicating Sequential Processes (CSP). This model describes the functionality in a very compact manner. Furthermore, it is possible to make profound statements about stability and security of the system.

The first step in our proposed design strategy is to create and test a functional CSP model. In a second step a designer has the flexibility to implement a CSP process in either hardware or software. On the system level this strategy leads to simpler and higher integrated designs, because it is possible to balance the workload between software and hardware without additional considerations about functionality or stability. Higher integration removes many of the stand alone components present in the current system design. So, the proposed system design has a lower component count, and hence a lower power consumption. This is true if we assume that: (a) each component requires some energy for housekeeping tasks like memory refreshes and (b) data exchange between individual components is less energy efficient than data exchange within an integrated system. Furthermore, most of the literature about higher integrated systems supports this claim [10].

We present in Section 1 the current, the desired and our proposed hardware for the IMS processing platform. Section 2 discusses the processing platform design. Software implementation aspects for the proposed processing platform are discussed in Section 3. In Section 4 we demonstrate how to implement a NoC using the design flow introduced in this paper. The paper closes with conclusions and further work in Section 5.

1. Proposed In-Situ Processing Platform

Before we propose a new processing platform for the IMS it is necessary to evaluate the existing systems from our partners in terms of storage and communications. This is good practice in order to prevent incompatibilities with existing structures.

1.1. Current In-Situ Monitoring System

A wide variety of electrical interfaces are used to connect sensor probes with the processing platform, ranging from analogue interfaces over RS-232 to USB. Similarly, the connection between IMS and Data Centre can be implemented using various different communication standards, but presently there is a bias towards using the mobile phone infrastructures, such as GSM (Global System for Mobile communications) [11]. As processing platform, the partners currently use a wide variety of processors ranging from off the shelf PC processors, over microcontrollers such as Intel 8051 or TI MSP430, to CPLDs (Complex Programmable Logic Devices) and FPGAs.

1.2. Desired In-Situ Monitoring System

The desired features of the next generation IMS include service intervals of 6 months and longer, smaller physical size of the system, and also more compatibility between sensor probes and processing platform is desired. Furthermore, our partners would like to have up to 20 sensor probes per in-situ measurement system. Additionally, one proposes to use IEEE 1451.x [12,13] to interface sensors with the processing platform. However, this standard is still in development and hence modifications on its interface are still possible. To prevent any loss of measurement data in case of a communication breakdown, the partners desire local data storage of up to 10 MB. To link IMS and Data Centre mobile phone infrastructure is still desired. However, a little flexibility on this side is still appreciated, especially with the background of GSM being phased out during the next 10 years and UMTS (Universal Mobile Telecommunications System) [14,15] taking over. Another interesting communication standard which can be used for this link is WiMax (Worldwide Interoperability for Microwave ACCess (IEEE 802.16)) [16,17], of which deployment has started in urban areas already. Taking all these points into consideration resulted in the processing platform for the in-situ measurement system we propose.

1.3. System Requirements

The biggest design constraint for the next generation in-situ measurement station is the service interval of 6 months and longer. This requires a very energy efficient measurement system design. One way to reduce the energy consumption is higher system integration. In the best case this results in a SoC (System on Chip) where sensors, processing platform, communication system, and battery are integrated in a single piece of silicon. Because all the components are specially designed for this SoC, unnecessary and power consuming abstraction layers can be avoided. A SoC can be seen as the ultimate long term goal. However, in the short term this solution has a number of drawbacks for the WARMER project:

- *High initial investment* – producing a SoC requires custom made components, i.e. ASICs (Application Specific Integrated Circuits) and the setup of specialised manufacturing lines. This does make sense for mass market products with high production volumes, but not for products like a measurement system of which only a few thousand units will be deployed across Europe.
- *Inflexible* – once production of systems has started, it is very hard to change anything. This makes the system not future secure: imagine what happens if GSM is finally replaced by UMTS and the SoC is designed to use GSM. The result is partial redevelopment. Similarly, we would be unable to utilise newly available sensor technology.

All these points make a SoC an unsuitable design approach for the desired in-situ measurement systems. However, one main point still holds:

A system on chip consumes less power because it avoids unnecessary abstraction layers.

For the proposed processing platform this means: we must find a way to avoid unnecessary abstraction layers and at the same time we need the ability to interface to different building blocks of the IMS.

1.4. Proposed Design

The flexibility of FPGAs permits us to accommodate all electrical interfaces used by our partners, ignoring any necessary voltage conversions for the moment. Furthermore, FPGAs can also perform signal processing tasks. Due to their truly parallel nature FPGAs can be clocked much slower, compared to machine-architectures, while performing the same processing. In general, higher clock speeds result in higher energy consumption. On the other hand there are some parts of the system, for instance the measurement scheduler, which are best implemented using a machine-architecture. Furthermore, having a built-in machine-logic lowers the barriers for our partners not used to FPGAs to use the system. To solve this problem we propose to utilise an FPGA with a built-in processor, such FPGAs are readily available from vendors such as Xilinx, Inc. Presently, we are evaluating a Xilinx Virtex 4 (XC4VFX12) [18] which has an embedded PowerPC 405. Figure 3 shows an overview of the hardware of the proposed practical solution.

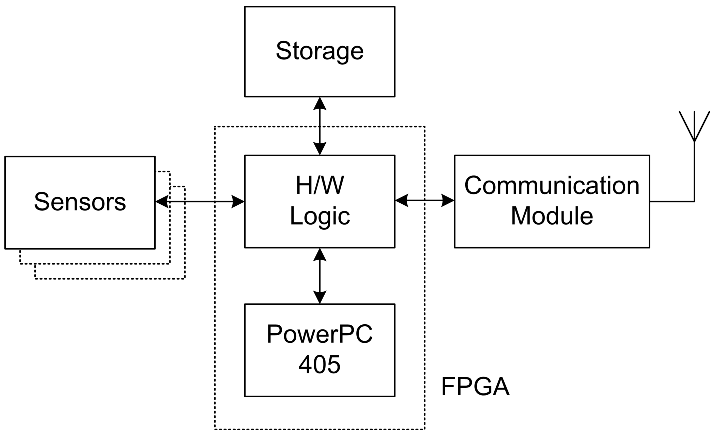

Figure 3. Hardware setup of the in-situ measurement station

The IMS hardware is a mix of hardware-logic and machine-logic, which must be combined in order to achieve our goal of a low power processing platform for the in-situ measurement system. Furthermore, this system is a true parallel system where new samples are aggregated, while older ones are analysed, and the results of previous analysis are sent to a Data Centre, at the same time. To avoid any race, deadlock, or livelock conditions we decided to follow the principles of CSP (Communicating Sequential Processes) and treat the different entities as CSP processes. This means, these entities only communicate over CSP style channels. What still remains to do is to design the interface between the different components, especially between the software executed by the machine-logic and the hardware-logic.

2. System Design Inside the FPGA

The previous section detailed the proposed hardware setup for the IMS. This is only the outside view of the system, what happens inside the FPGA, is much more interesting. Figure 4 shows one possible configuration / use of the FPGA. In the centre of the FPGA is a native PowerPC 405 core, which acts as central controller of the IMS, communicating over FDLs (Fast Duplex Links)[1] with the Sensor Controllers 1–N, Storage Controller, Comms

[1]A fast duplex link consists of two FSLs (Fast Simplex Links) [19], one for each direction.

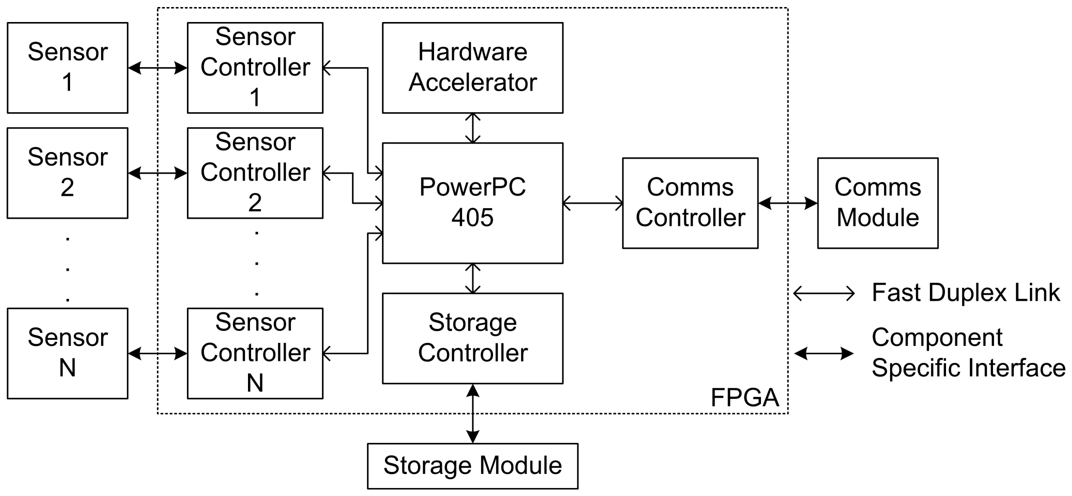

Figure 4. Inside view of the processing platform

Controller, and a Hardware Accelerator. In the following we detail the role of each of these components:

- Sensor Controllers – These interface the PowerPC core to the individual sensors. This interfacing involves not only protocol translation but also includes the electrical interface of the sensor. The PowerPC core communicates with the Sensor Controllers using a unified interface. This unified interface is a dedicated protocol between the Sensor Controllers and the PowerPC core. Furthermore, the Sensor Controllers can perform sensor specific signal processing. This avoids doing this processing in the sensor and hence allows a higher integration of the complete system.
- Comms Controller – The task of the Comms Controller is to interface the PowerPC core to a Comms Module. Similar to the Sensor Controller the Comms Controller performs not only a protocol translation but also provides the necessary electrical interface. The Comms Controller and the PowerPC core communicate over a FDL using a standardised protocol. This allows us to exchange the Comms Module, for instance to move from GSM to UMTS.
- Storage Controller – The Storage Controller abstracts the interface of the Storage Module and provides it in the form of a predefined protocol over a FDL.
- Hardware Accelerator – The Hardware Accelerator performs signal processing tasks which are too complex for the PowerPC core. The PowerPC core communicates with the Hardware Accelerator using a FDL. The system, shown in Figure 4, contains only a single Hardware Accelerator. However, there is no reason to limit the number of hardware accelerators in the system.

All these components are implemented as hardware-logic cores using the normal development tools. In the following we detail how the PowerPC core will be integrated in the design.

2.1. Integration of the PowerPC Core

The hardware-logic setup is fairly simple, each component represents one process, connected via FDLs to the PowerPC core. The controllers interface to hardware entities outside the processing platform, using hardware specific interfaces. Figure 5 shows the software process structure executed by the PowerPC. This structure is very similar to the hardware-process structure, detailed in Figure 4. In the center of the design is the *IMS Control Process*, which controls the IMS.

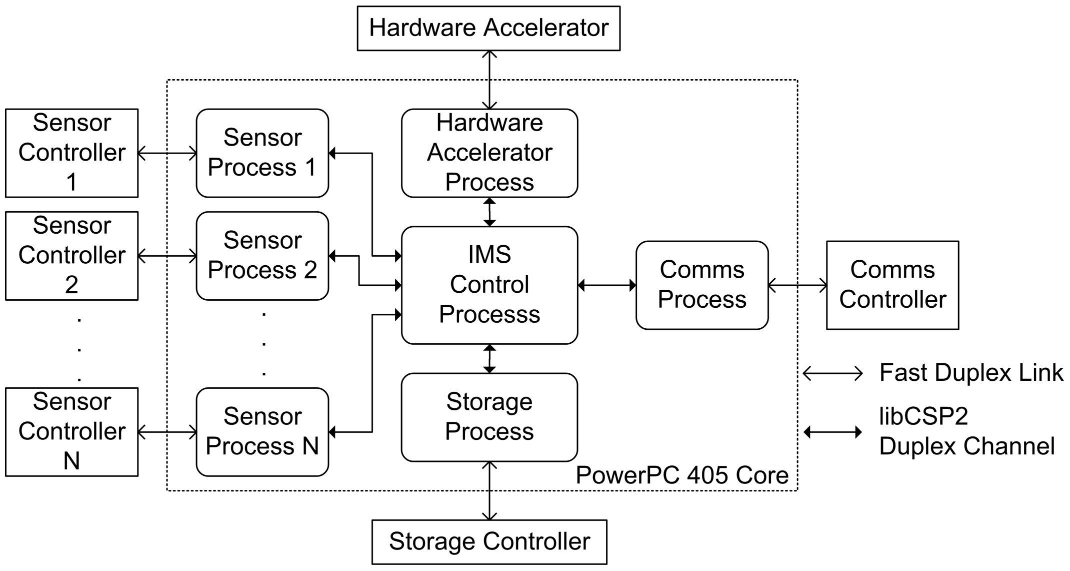

Figure 5. Process network within the PowerPC

Each present Sensor Controller is represented by a Sensor Process in software. This Sensor Process translates requests from the IMS Control Process into requests for the Sensor Controller, furthermore it may do additional signal processing. Possible applications for this sensor specific signal processing are, for instance: sensor calibration, detection of faulty sensors, or data type conversions.

The IMS Control Process communicates with the Comms Controller via the Comms Process. This process performs the translation, and contains all necessary configuration / authentication information for the chosen communication network. For instance when using GSM to connect to the Data Centre, the authentication to the GSM network provider is handled by this process. Furthermore, this process handles the identification of the IMS with the Data Centre, and ensures that no messages are lost between IMS and Data Centre.

To communicate with the Storage Controller the IMS Control Process communicates with the Storage Process. Initially, this process will only perform a simple request translation. However, in future we can add a file system like functionality, for instance by appending the current time and date to each entry. Furthermore, it could support data encryption to prevent others reading the stored information.

Finally, the Hardware Accelerator Process translates requests between the IMS Control Process and the Hardware Accelerator. This process will expose a call-channel interface. This allows users to utilise the functions the Hardware Accelerator offers just like a normal function call.

3. Implementation of the Software

The previous sections detailed the proposed structure of the IMS processing platform. A vital aspect of the proposed system is the duality between hardware and software. In the following we discuss how we plan to implement the software system run by the PowerPC core. There are a number of constraints which we need to take into consideration when implementing the software:

- Amount of memory available to the PowerPC; The currently proposed FPGA (XC4VFX12 [18]) offers 81KB of BRAM (Block RAM) memory, i.e. if any hardware-logic requires memory, this memory is deducted from these 81KB. It is of course possible to use external RAM, however this requires more energy, more space on the PCB (Printed Circuit Board), and finally it costs more money. Another possibility is to choose an FPGA with more internal memory, but even then the memory footprint of the software remains an issue.

- Utilisation of non-standard interfaces of the PowerPC; The proposed processing platform relies on the use of FDLs to communicate with the hardware-logic. Hence, the chosen operating system must allow us to access the FDL interface provided by the PowerPC core.
- The choice of programming languages is limited; We are not the only ones developing software for the proposed processing platform. In fact most of the extensions to it will be developed by our partners in the consortium. The questionnaire revealed that most people are familiar with C and C++. It is safe to assume that they have already legacy code which they would like to reuse.

3.1. Operating System

In order to comply with these constraints, we decided to use XMK (Xilinx Microkernel) [20] as OS (Operating System). XMK is a small OS for the Xilinx MicroBlaze SoftCPU and hard wired PowerPC 405 cores. This OS abstracts the access to the FDL interfaces for both MicroBlaze and PowerPC 405.

After choosing an applicable OS it is time to choose a CSP environment to implement our processes on the machine-architecture. One possible choice is to port and extend the Transterpreter [21,22] to our chosen hardware platform. Choosing the Transterpreter implies that the software will be developed in Occam. While this is no problem for us, our partners don't have that background and furthermore they have legacy code in other languages such as C. Therefore, we need a C based solution. We therefore propose to use libCSP2 [23] as CSP environment for the processing platform. There are two reasons for this: firstly, libCSP2 has already built-in support for FDLs, secondly it allows one to develop CSP style software in C. This ensures a flat learning curve for our partners, when they want to develop their own extensions to our system.

3.2. Recent Developments of libCSP2

There have been a number of small enhancements to libCSP2 recently. First, libCSP2 now abstracts FSLs as normal channels, i.e. a process can not determine whether it uses a software channel or an FSL based channel. This allows developers greater freedom when doing multi-core designs with libCSP2.

Furthermore, we changed the build system from autotools to CMake [24]. This step allows users now to build the library outside of their current software project. This means the libCSP2 source code is not present in the users software project.

Presently, we are working on a formal verification of the implementation of libCSP2 on XMK. However, this work will still take some time to complete.

4. Example: Sensor Integration

This section demonstrates the proposed CSP based design flow for NoCs works. Our design goal is the integration of a new sensor into the processing platform. The desired functionality is straightforward: the IMS Control Process acquires a measurement value form a sensor and triggers an alarm is the acquired measurement value is above a certain threshold. We implement this simple example with libCSP2 technology on a Virtex-4 FPGA.

The first step in the design is to create and test a functional CSP model of the system. Subsequently, the CSP processes are mapped onto the available processors, in this case FPGA and PowerPC. After the mapping step follows the implementation. The individual processes are implemented using processor specific tools and networked using domain specific channels. The following text concentrates on the implementation of the processes on the PowerPC using libCSP2 and how to link them with processes located within the FPGA.

4.1. CSP Model

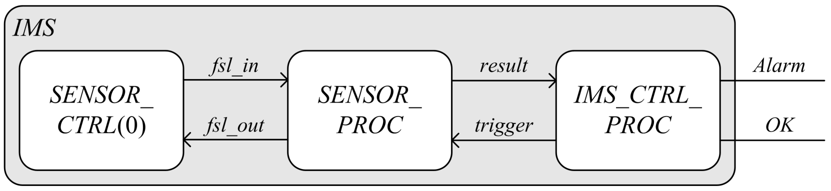

Figure 6. IMS process network

Figure 6 shows the *IMS* process network which represents the system functionality. The main task of the process network is to raise an *Alarm* when a sensor detects a harmful pollution. We model this functionality with three processes: *SENSOR_CTRL*, *SENSOR_PROC*, and *IMS_CTRL_PROC*. These communicate over the channels: *fsl_in*, *fsl_out*, *trigger*, and *result*. We did not do extensive tests on the CSP model, because it is very simple. The following paragraphs explain the CSP model for each of these processes.

The process $SENSOR\_CTRL(i)$ (Equation 1) represents the Sensor Controller. This process expects to receive the command value 48^2 from the channel *fsl_out* and then returns a measurement value in the range $[0..49]$. Any other value on channel *fsl_out* will be ignored.

$$\begin{aligned} &SENSOR\_CTRL(i) = \\ &\quad fsl\_out?x : \{48\} \rightarrow fsl\_in!(i \bmod 50) \rightarrow SENSOR\_CTRL(i+1) \end{aligned} \tag{1}$$

Equation 2 specifies the process *SENSOR_PROC* which represents the Sensor Process. The process waits for any message on channel *trigger* and then requests a measurement value from the Sensor Controller, by sending the value 48 over channel *fsl_out*. Then it waits for a message from the Sensor Controller on channel *fsl_in*. This message represents the measurement value. The process then relays this value to the IMS Control Process over the channel *result*. The process is now ready to process the next request.

$$\begin{aligned} &SENSOR\_PROC = \\ &\quad trigger?x \rightarrow fsl\_out!48 \rightarrow fsl\_in?x \rightarrow result!x \rightarrow SENSOR\_PROC \end{aligned} \tag{2}$$

Process *IMS_CTRL_PROC* (Equation 3) represents the IMS Control Process. The IMS Control process requests a measurement value from the Sensor Process, by sending a message over the channel *trigger*. After that it waits for the measurement value to arrive on the channel *result* and then compares the received value with 42. If the measurement value is smaller or equal 42 everything is OK and the process issues an *OK* event. Otherwise, the process issues an *Alarm* event. In both cases the process recurses to start a new round of measurement.

[2]The command value, as well as the range of measurement values ($[0..49]$) and the threshold value (42), are arbitrary chosen values.

$$IMS\_CTRL\_PROC =$$

$$trigger!1 \rightarrow result?x \rightarrow \begin{cases} Alarm \rightarrow IMS\_CTRL\_PROC & \text{if } x > 42 \\ OK \rightarrow IMS\_CTRL\_PROC & \text{otherwise} \end{cases} \quad (3)$$

Process *IMS* (Equation 4) represents the complete IMS, which consists of the processes: *SENSOR_CTRL*(0), *SENSOR_PROC*, and *IMS_CTRL_PROC*. To avoid any outside interference all transactions on channels: *fsl_in*, *fsl_out*, *trigger*, and *result* are hidden. Only the events *Alarm* and *OK* are visible to the outside world.

$$\begin{aligned} IMS = {} & SENSOR\_CTRL(0) \parallel SENSOR\_PROC \parallel IMS\_CTRL\_PROC \\ & \setminus \{| \, fsl\_in, fsl\_out, trigger, result \, |\} \end{aligned} \quad (4)$$

4.2. Mapping the Processes onto the Available Processors

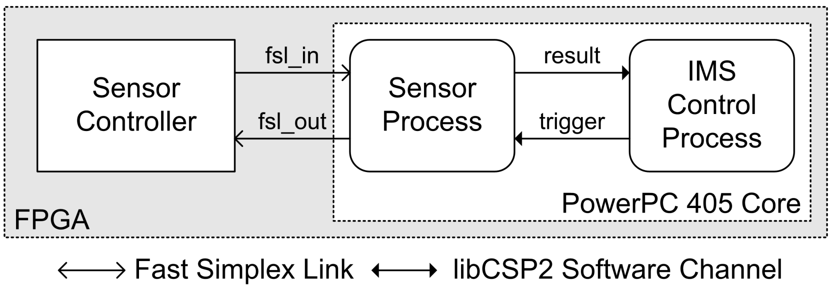

Figure 7. Process mapping onto the processors of the processing platform

Figure 7 illustrates the mapping of the CSP processes onto the processors of the processing platform. The Sensor Controller gets implemented outside the PowerPC, in order to interface directly with the Sensor. The Sensor Process is located within the PowerPC core together with the IMS Control Process. The Sensor Controller and the Sensor Process communicate using FSLs (Fast Simplex Links), while Sensor Process and IMS Control Process use libCSP2 software channels.

4.3. Implementing the Processes Located within the PowerPC

Listing 1 is the libCSP2 implementation of process *SENSOR_PROC* (Equation 2).

```
void SensorProcess(pChannel trigger, pChannel result, \
                   pChannel fsl_in, pChannel fsl_out){
  int msg = 0;
  while(1){
    ChanInInt(trigger, &msg);
    ChanOutInt(fsl_out, 48);
    ChanInInt(fsl_in, &msg);
    ChanOutInt(result, msg );
} }
```

Listing 1: Sensor Process Implementation using libCSP2

Listing 2 gives the libCSP2 implementation of process *IMS_CTRL_PROC* defined in Equation 3. From a functional point of view, the main difference between the functional model and the implementation lies in the handling of the alarm. The implementation does not issue an alarm event over a channel, instead it outputs the corresponding strings onto the console, (lines 7 and 9).

```
void ControlProcess(pChannel trigger, pChannel result){
  int value = 0;
  while(1){
    ChanOutInt(trigger, 1); // trigger a new measurement value
    ChanInInt(result, &value); // receive the new value
    if(42 < value){
      xil_printf("Alarm\r\n");
    }else{
      xil_printf("OK\r\n");
} } }
```

Listing 2: IMS Control Process Implementation

The processes located in the PowerPC core have now to be instantiated, linked with the Sensor Controller and then executed. Listing 3 demonstrates how this is done using libCSP2. The listing consists of three sections: one declaring necessary variables (lines 2 – 7), a definition section (lines 10 and 11). The last section is the function `void* shell_main(void)` which uses these declarations and definitions (lines 13 – 21).

```
   //Channel and process declarations;
 2 pProcess sensor  = NULL;
   pProcess control = NULL;
 4 pChannel trigger = NULL;
   pChannel result  = NULL;
 6 pChannel fsl_in  = NULL;
   pChannel fsl_out = NULL;
 8
   // Defining necessary intermediate functions;
10 void procSensor(void) {SensorProcess(trigger, result, fsl_in, fsl_out);}
   void procControl(void) {ControlProcess(trigger, result);}
12
   void* shell_main(void*  dummmy){
14   CSP_ChanAllocInit( &trigger, CSP_ONE2ONE_CHANNEL);
     CSP_ChanAllocInit( &result, CSP_ONE2ONE_CHANNEL);
16   FSL_AllocInitChannelInput( &fsl_in, CSP_ONE2ONE_CHANNEL, 0);
     FSL_AllocInitChannelOutput( &fsl_out, CSP_ONE2ONE_CHANNEL, 0);
18   ProcAllocInit(&sensor, procSensor);
     ProcAllocInit(&control, procControl);
20   ProcPar(control, sensor, NULL);
   }
```

Listing 3: IMS Processing Platform Setup

Lines 10 and 11 define intermediate functions, which represent the Sensor Process and the IMS Control Process. The reason for these intermediate functions is that libCSP2, in its current state, only allows un-parametrised functions to act as processes. Unfortunately, the functions which represent the Sensor Process and the IMS Control Process have parameters, making these intermediate functions necessary.

The function `void* shell_main(void)` (line 13) represents the program entry point. Once started it allocates and initialises the channels: `trigger` and `result` as normal software channels (lines 14 and 15). To connect the Sensor Process with the Sensor Controller the function allocates and initialises two FSL channel-ends: `fsl_in` and `fsl_out`. The channel-end `fsl_in` gets allocated as FSL channel input for FSL-ID 0 (line 16). This means that the process using this channel end may only input data from the FSL, but not output data to it. The function then allocates and initialises the channel-end `fsl_out` as FSL channel output for FSL-ID 0 (line 16). The statements that follow (lines 18 – 20), allocate and initialise the two processes and then execute them in parallel. This completes the implementation of the process network located in the PowerPC.

This example demonstrated how to implement NoCs using libCSP2 and Xilinx FSL. Furthermore, it supports our claim of unification of channels within libCSP2. To move the Sensor Process outside the PowerPC no change of the IMS Control Process is necessary, only the software channels `trigger` and `result` have to be replaced with FSL channel-ends.

5. Conclusions and Further Work

This paper proposed a processing platform design for the WARMER in-situ monitoring system. One of the requirements of this in-situ monitoring system is: Long service intervals in the range of 6 to 12 months. Another aspect is the flexibility to work with or replace various existing systems used by other WARMER collaborators. One last requirement is to design the system such that our partners can reuse their code and extend the system without our help. In the practical part of the paper we demonstrated how FPGA technology can be used to achieve higher system integration with more flexibility. These goals were achieved with a network of software and hardware processes.

A big advantage of the proposed system is the sheer ease with which it allows the designer to create hybrids of hardware- and machine-logic, when using CSP style communications between these processes. Another benefit of Communicating Sequential Processes is the duality of hardware- and machine-logic. Each hardware-logic core has a process representation within the machine-logic. This allows the designer to choose the processing platform which executes specific data or control centric algorithms. This freedom leads to optimised systems, because of an optimal use of processing resources. Furthermore, it is easy to follow the data flow within the system. This makes the system easy to understand and extend. The use of libCSP2 as CSP environment for the software part of the system allows the partners to reuse previously developed algorithms without too much difficulty.

The approach we present in this paper is not restricted to processing platforms embedded in in-situ monitoring systems but is generally applicable to hardware-software co-design.

5.1. Further Work

This project is still in the drafting stage, and there is still a lot of work to be done. Nevertheless, we already see a number of areas to be explored. In its current state the processing platform needs to be designed / compiled specifically for the used sensors. While this is fine for prototyping and small scale use it becomes a nuisance once the system is deployed out in the field, because it is not possible to plug and play the sensors. To solve this, two areas need to be investigated: partial reconfiguration of the FPGA, and partial process network reconfiguration of the libCSP2 process network. Here we see again the duality of the system design, where hardware- and machine-logic are closely coupled. For libCSP2 these requests mean to implement stateful poisoning. Furthermore, libCSP2 needs an extension that provides call-channels. However, the previously mentioned further work items are long term work items, in the short term we need to start developing the protocols used between the PowerPC 405 core and the hardware-logic cores. Not to forget convincing our partners of the advantages of this design approach.

Acknowledements

This work was supported by the European FP6 project “WARMER” (contract no.: FP6-034472).

References

[1] Amara Gunatilaka. Groundwater woes of Asia. *Asian Water*, pages 19–23, January 2005.

[2] Amara Gunatilaka. Can EU directives show Asia the Way. *Asian Water*, pages 14–17, December 2006.

[3] Results of the IST-2000-28084 Project SEWING: System for European Water monitorING. Available (23.04.2007) at: http://www.sewing.mixdes.org/downloads/final_results.pdf, December 2004.

[4] Renata Mamińska and Wojciech Wróblewski. Solid-state microelectrodes for flow-cell analysis based on planar back-side contact transducers. *Electroanalysis*, 18(13–14):1347–1353, July 2006.

[5] V. V Malinovsky and S. Sandven. SAR monitoring of oil spills and natural slicks in the Black Sea. Submitted to *Remote Sensing of Environment*., 2007.

[6] A Legin, A Rudnitskaya, B Seleznev, and D Kirsanov. Chemical sensors and arrays for simultaneous activity detection of several heavy metal ions at low ppb level. In *Proceeding of Pittcon 2004*. Pittsburgh Conference, March 2004.

[7] M. T. Castañeda, B. Pérez, M. Pumera, A. Merkoçi, and S. Alegret. Sensitive stripping voltammetry of heavy metals by using a composite sensor based on a built-in bismuth precursor. *Analyst*, 130(6):971–976, 2005.

[8] Luca Benini and Giovanni De Micheli. Networks on Chips: A New SoC Paradigm. *Computer*, 35(1):70–78, 2002.

[9] Grant Martin. Book Reviews: NoC, NoC ... Who's there? *IEEE Design and Test of Computers*, 23(6):500–501, 2006.

[10] Hugo De Man. System-on-Chip Design: Impact on Education and Research. *IEEE Des. Test*, 16(3):11–19, 1999.

[11] M. Rahnema. Overview of the GSM System and Protocol Architecture. *IEEE Communications Magazine*, 31(4), April 1993.

[12] Richard D. Schneeman. Implementing a standards-based distribution measurement and control application on the internet. Technical report, U.S. Department of Commerce, Gaithersburg, Maryland 20899 USA, June 1999.

[13] James D. Gilsinn and Kang Lee. Wireless interfaces for IEEE 1451 sensor networks. In *Proceedings of the First ISA/IEEE Conference*, pages 45–50. IEEE, November 2001.

[14] Antonios Alexio, Dimitrios Antonellis, and Christos Bouras. Evaluating Different One to Many Packet Delivery Schemes for UMTS. In *WOWMOM '06: Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, pages 66–72, Washington, DC, USA, 2006. IEEE Computer Society.

[15] Xiao Xu, Yi-Chiun Chen, Hua Xu, Eren Gonen, and Peijuan Liu. Parallel and distributed systems: simulation analysis of RLC timers in UMTS systems. In *WSC '02: Proceedings of the 34th conference on Winter simulation*, pages 506–512. Winter Simulation Conference, 2002.

[16] S.J. Vaughan-Nichols. Achieving wireless broadband with WiMax. *Computer*, 37(6):10–13, June 2004.

[17] Teri Robinson. WiMax to the world? *netWorker*, 9(4):28–34, 2005.

[18] Xilinx, Inc., 2100 Logic Drive San Jose, CA 95124-3400, United States of America. *Virtex-4 Family Overview*, DS12 (v2.0) edition, January 2007.

[19] Xilinx, Inc. *Fast Simplex Link (FSL) Bus (v2.00a)*, 1 December 2005.

[20] Xilinx, Inc, 2100 Logic Drive San Jose, California 95124 United States of America. *OS and Libraries Document Collection*, 24 October 2005.

[21] Christian Jacobson and Matthew C. Jadud. The Transterpreter: A Transputer Interpreter. In Ian R. East, David Duce, Mark Green, Jeremy M. R. Martin, and Peter H. Welch, editors, *Communicating Process Architectures 2004*, pages 99–106, September 2004.

[22] Damian J. Dimmich, Christian Jacobson, and Matthew C. Jadud. Native Code Generation using the Transterpreter. In Frederick R. M. Barnes, Jon M. Kerridge, and Peter H. Welch, editors, *Communicating Process Architectures 2006*, pages 269–280, September 2006.

[23] Bernhard Sputh, Oliver Faust, and Alastair R. Allen. Portable CSP Based Design for Embedded Multi-Core Systems. In Frederick R. M. Barnes, Jon M. Kerridge, and Peter H. Welch, editors, *Communicating Process Architectures 2006*, pages 123–134, September 2006.

[24] Ken Martin and Bill Hoffman. *Mastering CMake 2.2 Edition*. Kitware, Inc., Clifton Park NY, USA, 24 February 2006.