PROGRAMMABLE HARDWARE ARCHITECTURES FOR SENSOR VALIDATION

M. P. Henry[†*], N. Archer[†], M. R. A. Atia[†], J. Bowles[†], D. W. Clarke[†], P. M. A. Fraher[†], I. Page[‡], G. Randall[‡], J.C-Y. Yang[†].

[†] *Department of Engineering Science, Parks Road, Oxford OX1 3PJ.*
[‡] *Computing Laboratory, Parks Road, Oxford. OX1 3QD.*
[*] *To whom correspondence should be addressed: manus.henry@eng.ox.ac.uk*

Abstract. A previous paper (Henry, 1995a) introduced the technique of hardware compilation as the basis for developing highly flexible programmable hardware platforms for control applications such as sensor validation. This paper describes two PC-hosted architectures for sensor validation research. The first holds up to two FPGAs and supports a daughter board with application-specific circuitry. The second is based on the transputer TRAM standard, and consists of programmable hardware modules providing interfacing and low-level signal processing between the transputer and arbitrary I/O components. Three applications are described, based upon a thermocouple, a dissolved oxygen probe and a Coriolis mass flow meter.

Key Words. Field-Programmable Gate Arrays (FPGAs); hardware compilation; Handel; occam; transputers; sensor validation; uncertainty; fault detection and diagnosis; fieldbus; thermocouple; dissolved oxygen sensing; Labview.

1. INTRODUCTION

This paper describes two hardware architectures developed for control applications as part of the "Valcard" project, a collaboration between the Computing Laboratory and the Engineering Science Department at Oxford University. The particular application area considered is sensor validation, and three examples are given of the architectures in use; however, many of the requirements of the particular domain are similar to those of a broader class of control applications.

The principal enabling technology is the Field Programmable Gate Array (FPGA), which is a configurable matrix of simple logic cells. A technique called hardware compilation (Page and Luk, 1991) is used to specify FPGA functionality in a high-level imperative programming language called Handel, which is a derivative of **occam** (Inmos, 1988a). The background to the Valcard Project, including introductions to FPGAs, hardware compilation, and the requirements for hardware platforms for sensor validation, are described in (Henry, 1995a).

This paper provides a brief review of these topics, and then describes the two PC-hosted architectures. The first card, aimed at simpler applications, consists of an ISA-bus card which can host up to two FPGAs, and which can also support a daughter board holding application-specific circuitry. The second design is modular, based on the transputer (Inmos, 1988b), in which FPGA modules provide interfacing between the transputer and application-specific circuitry, and may in addition carry out low-level signal processing to reduce the computational burden on the processor.

After each architecture is described, applications are presented. The first, the so-called FPGA card, has been used in thermocouple and dissolved oxygen sensor applications, while the second, the transputer card, has been applied to a Coriolis mass flow meter.

These architectures provide the advantages of a software-based prototyping environment: high-level abstract design definition, automated design transformation, symbolic debugging, and a high degree of re-use between applications. At the same time, few constraints are placed on the hardware functionality implemented using the cards. Indeed, as much of the hardware is described in software, experimentation with alternative designs, and continuous revision and updating, are rendered far easier than with traditional hardware design approaches, and so a wider range of hardware solutions might be considered using this route.

Finally, the limitations of these architectures are considered along with issues for further research. Key themes are the extension of the design methodology from prototyping to the manufacture of ASICS, and the provision of Fieldbus communication capability.

2. CONTEXT AND MOTIVATION

This section explains the context of and motivation behind the Valcard project. A summary of the sensor validation research aims and prototyping requirements is followed by brief notes on FPGAs and hardware compilation. Alternative solutions are considered, and then the Valcard project is summarised. Each of these topics is covered in more detail by (Henry, 1995a).

2.1. Sensor validation

For a number of years the Control Engineering group at Oxford has been developing the concept of the self-validating or SEVA sensor (Henry and Clarke, 1993). The term "sensor" includes both the *transducer(s)* and what is often called in industry the *transmitter*, which converts the transducer signals into a form (pneumatic pressure, current, or more recently digital message) which can be sent to the control system. In the SEVA context, it is assumed that the sensor has local computational power, and a digital communication capability which enables it to send and receive messages of arbitrary complexity.

Like many intelligent sensors available commercially, the SEVA sensor performs self-checking. Instead of outputting simple device-specific error codes or validity bits, it generates generic validity metrics for each measurement (Fig. 1):

- The validated measurement value (VMV) is the best estimate of the true measurand value, taking all diagnostic information into account. If a fault occurs, then the VMV is corrected to the best ability of the sensor. In the most severe cases (where the raw data are useless or have stopped completely), the current VMV is extrapolated from past measurement behaviour.
- The validated uncertainty (VU) is the uncertainty associated with the VMV. The metrological definition is used here (Kline and McClintock, 1953; ANSI, 1985): the VU gives a confidence interval for the true value of the measurand. For example, if VMV is 2.51 units, and the VU is 0.08, then there is a 95% chance that the true measurement lies within the interval 2.51 ± 0.08 units. The VU takes into account all likely sources of error, including noise, measurement technology and any fault-correction strategy currently being used.
- The measurement value status is a discrete-valued flag indicating how the VMV has been calculated. A few basic categories are admitted, such as CLEAR - the VMV has been calculated normally, BLURRED - the VMV is still based on live data, but is being corrected for a fault, and BLIND - the VMV is being projected from past behaviour. The MV status allows users of the measurement to determine whether it is acceptable - for example, BLIND data should never be used for feedback control.

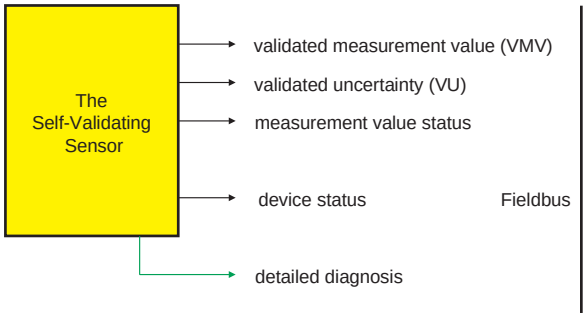

Fig. 1. Parameters generated by the Self-Validating Sensor

It is becoming increasingly common for sensors to generate more than one measurement (e.g., process temperature and pressure), a practice encouraged by digital communications, whereby more than one data channel can be transmitted over a single pair of wires. In the SEVA scheme the VMV, VU and MV status are generated for *each* measurement (as, for example, a single fault may affect each measurement in a different way). In addition, a single, generic device status is provided to summarise the health of the physical instrument, and detailed diagnostics are made available as required.

The SEVA group has several active research areas:

- The development of prototype SEVA sensors such as Coriolis mass flow (Henry, 1994a), thermocouple (Yang, 1993), and dissolved oxygen (Clarke and Fraher, 1995).
- Self-validating actuators, such as valves (Alsop, 1995).
- Control, fault detection and validation of loops and processes which use SEVA sensors and actuators (Clarke, 1995; Yang and Clarke, 1995).

For experimental work and practical demonstrations, prototype SEVA instruments are a prerequisite. Such prototypes have usually been based upon commercial instruments. In the first generation of SEVA sensors a PC was interfaced to an industrial sensor to pick up signals relating to measurement and diagnostics (Fig. 2). It is frequently a further requirement for control signals to pass from the PC to the sensor, for example to activate testing circuitry. Often, the transmitter is bypassed entirely, and the PC is interfaced directly to the transducer. The PC acts as a self-validating transmitter by performing fault detection, validation, and measurement and uncertainty calculations. The net effect is that the PC and sensor together provide an on-line demonstration of how a standalone SEVA transmitter would behave.

Although there are several advantages obtained by basing prototypes upon existing industrial instruments

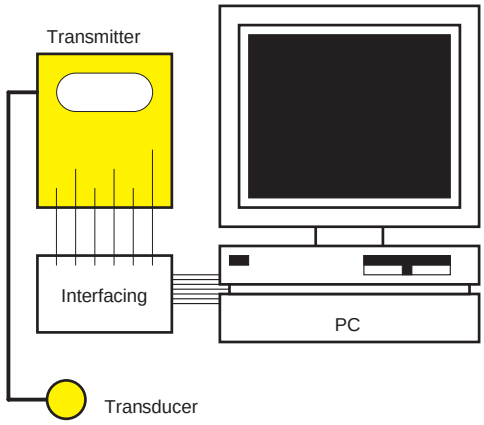

Fig. 2. Typical hardware for SEVA application

(results are more readily exploited, and are taken more seriously by users), a disadvantage is that access to the desired signals is not readily available, as the devices were not developed with such provision in mind.

The development of a sensor-specific interface to the PC is an intensive task, requiring perhaps several worker-months of effort, with regular liaison with the sensor design team. This work is frequently on the critical path of the entire instrument validation process, because the development (or confirmation) of diagnostic methods depends upon having access to the relevant signals so that their behaviour may be observed in a variety of faulty and non-faulty conditions.

As research proceeds the need for extensions or modifications to the hardware interface may become apparent (e.g., the introduction of additional active testing or on-line calibration procedures, or the monitoring of extra signals). It is therefore desirable for the hardware and signal conditioning portions, and thus their interface to the validating microprocessor, to exhibit a high degree of flexibility. Ideally the hardware should be as readily changeable as the software running on the processor, which can be recompiled and loaded with ease.

2.2. Alternative technologies

A number of technologies have been considered for carrying out prototyping work:

- **PC + commercial cards.** This was the approach used for developing the first generation of SEVA prototypes. In practice, it has been found that a general-purpose data-acquisition card rarely provides exactly what is needed for each SEVA application. Additional, customised circuitry is inevitably required, and such hardware is relatively inflexible when design modifications are required. Another limitation is that advances in, for example, ADC and DAC chip design (e.g. precision and/or bandwidth improvements) become available in general-purpose cards at a much later date than if it is possible to use the chips themselves directly.
- **Fixed architecture for sensor applications.** An alternative approach is to develop or buy a fixed architecture aimed at sensor applications. An example is the universal sensor interface IC (USIC) (Wilson *et al.*, 1996) developed as part of the EUREKA project JAMIE (Joint Analogue Microsystems of Europe). The USIC features a RISC processor, on-chip op-amps, 2 ADCs and DACs and RS-485 interfacing capability. Such developments offer an excellent low-cost route for the development of conventional sensors in a number of classes, but are insufficiently flexible to satisfy the broader requirements for validation. For example, the hardware and interfacing requirements for Coriolis validation listed in section 5 are far broader than those envisaged in the design of devices such as USIC.
- **Programmable hardware.** Here the hardware itself is programmable, so that modifications can be made as readily as to the validation software. Clearly, there are limits to the programmability of certain hardware components (e.g., analogue signal conditioning); the adoption of a modular structure would reduce the requirements for hardware change to a minimum. This approach allows the construction of a prototype using exactly the components (ADCs, DACs etc.) that might be used in a commercial system, and as such represents a step beyond the use of general-purpose data-acquisition cards. Furthermore, the choice of components is not restricted to those employed in such cards.

The Programmable Hardware approach was adopted as offering the maximum flexibility. The key hardware and software technologies required are now described.

2.3. Field Programmable Gate Arrays

A Field-Programmable Gate Array is an electronic chip containing an array of computing blocks (called configurable logic blocks or CLBs), each of which can perform a limited number of logical operations. An FPGA also contains a number of I/O blocks (IOBs), providing an interface between each external package pin and the internal logic. The function of each block, and the interconnections between them, are determined by on-chip static memory, which is configurable. Typically the FPGA is configured once on power-up from an EPROM or a processor, but it is also possible to repeatedly reconfigure the FPGA to perform a series of different functions.

Until recently there have been two categories of FPGA design tools. The first is the schematic drawing package, e.g., Workview (Viewlogic, 1991). The second is the behavioural description approach, in which requirements are specified using a variety of textual forms, such as declarations (e.g., pin assignments), Boolean equations, truth tables, and state-machine descriptions. VHDL (IEEE, 1994) is perhaps the most

important example of this approach, although it is not restricted to FPGA design. More recently, a third tool class has emerged, in which FPGA functionality is described in the form of an executable program (the hardware compiler).

The output of most design tools is a net-list consisting of the desired logic functions and the connections between them. The computationally-expensive task of place and route maps the net-list onto the resources available in the target FPGA, and generates a bit file to configure the device to perform the required task.

2.4. Hardware compilation

In hardware compilation (Hoare and Page, 1994), a programming language, or more exactly a specification language that can be executed, is used to describe FPGA functionality. The Handel language (Page and Luk, 1991, Spivey and Page, 1993) is based on **occam**. Its provision of high- granularity parallel and sequential instructions is particularly suitable for the specification of FPGA functionality. A description of the language is is provided by Spivey and Page (1993), while Henry (1995a) discusses in some detail the suitability of the language for defining FPGA functionality. A short example of Handel code, providing an interface and test output to a dual-channel 12 bit DAC, is shown in Appendix 1. The Handel compiler transforms a Handel declaration into a net-list, which is passed to the place and route process.

There are a number of advantages to using hardware compilation, as opposed to the traditional methods of FPGA design. Handel can satisfactorily express a whole range of requirements, from simple signal connection through to floating point operations, in a compact, textual form. Handel has well-defined semantics and so is amenable to analysis (e.g., for correctness) and automated generation. Through the provision of functions and parameterisation (Page, 1994), a high degree of reuse can be achieved between applications.

2.5. The Valcard project

This project was established to improve the technology used for implementing SEVA prototypes. The principal aim was the creation of an FPGA-based validation card, to be configured using hardware compilation techniques, with sufficient flexibility to be applicable to a wide range of instruments.

Commercial systems are available allowing rapid prototyping of digital systems using FPGAs (Guccione and Gonzalez, 1995 [1]). A range of platforms are supported, including PC, Sun and the transputer. The intended application domains include super-computing, multimedia, video processing and microprocessor prototyping, and the cost and (over-) specification of such systems are not well suited to the process instrumentation domain.

The main benefits of developing a programmable hardware platform for validation were expected to be the minimisation of application-specific hardware (as opposed to Handel and other software), and the maximal re-use of code between applications:

- FPGAs can implement a range of requirements from simple connections, to low-level signal conditioning and processing, which might otherwise be carried out by application-specific circuitry.
- The latest generation of ADCs and DACs, called ΣΔ devices (Boser and Wooley, 1983), have a variety of features to minimise analogue signal conditioning requirements, e.g. over-sampling, to eliminate the need for anti-aliasing, and programmable gains. Their correspondingly complex programmable interfaces are readily dealt with by Handel routines. Libraries of drivers can be reused in different applications.
- The provision of a card standardises aspects of all applications (e.g., using ISA bus and software ports to communicate data to PC-based software, power supply from PC bus).
- The first generation of SEVA sensors used a custom-built C++ library (called Siamese) combining host functionality (e.g., graphics) with instrument functionality (e.g., diagnostic reasoning). If the card has on-board processing power, it would be possible to split instrument and host functionality, thus aiding code development and re-use.

The first requirement of the project was to draw up the detailed specification for the card. Consideration of the application domain - industrial measurement - pointed to two separate philosophies:

- The most common forms of commercial sensor are powered via the same pair of wires used for data communication. In the 4-20mA domain this places a severe restriction on power consumption, and whereas with the advent of fieldbus more power may be available, this is more than compensated for by the extra power requirements to provide the processing necessary to operate the communication software. So, “powered-off-the-bus” or “two-wire” devices tend to be power- and cost-limited.
- The second class of commercial instruments enjoy independent power supplies (either mains or battery). Typically, these are devices which require extra power to carry out the measuring process (e.g., Coriolis, magnetic flow meters). In this smaller market segment, functionality is more important than power consumption.

1 Steve Guccione maintains a list of FPGA-based computing machines on the World-Wide Web: hhtp:/www.io.com/~guccione/HW_list.html

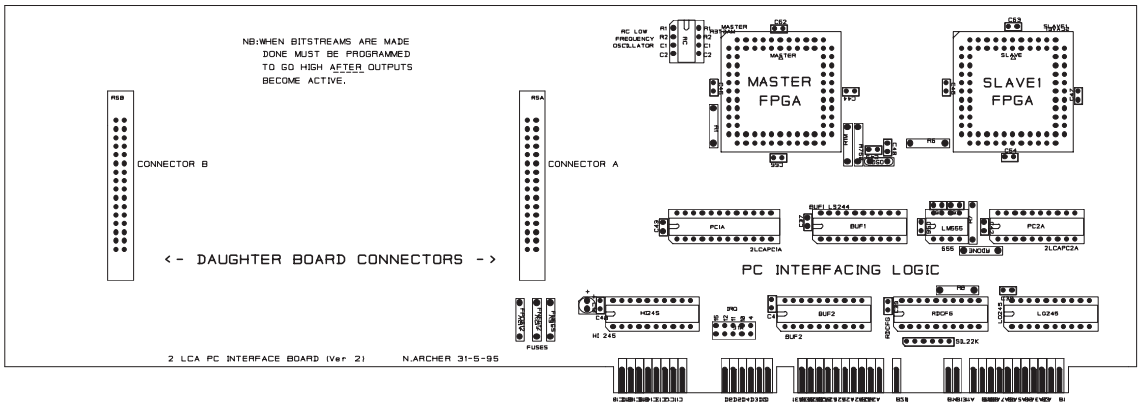

Fig. 3. Simplified structure of the FPGA card.

These two different approaches are reflected in the two architectures which were developed within the project.

The remainder of this paper describes these two architectures in more detail. The corresponding changes in application and host software are also outlined. Three sensor validation applications are presented which have been developed using the new cards. Finally, conclusions are drawn about the suitability of the architectures and proposals are made for improvements.

3. THE FPGA CARD

The FPGA card has been designed for maximum flexibility. It carries no microprocessor, but holds one or two Xilinx 3000 series FPGAs. It is intended either for very low-computation applications, in which case full validation can be programmed into the FPGAs, or simply to provide flexible interfacing for validation calculations taking place on the host PC. It has already been used for other applications, such as high-speed decoding in a communications project.

The FPGA card supports a range of Xilinx chips from the 3064 to the 3195, so the capacity and expense of the card can be matched to the application. An 84-pin PLCC package is used. A schematic of the card is shown in Fig. 3, while Fig. 4 shows a photograph of the card together with daughter boards for the thermocouple and dissolved oxygen applications.

3.1. Internal structure

The FPGA card has a 16-bit bus which is shared by the two FPGAs and the daughter board (Archer and Bowles, 1995). The bus connects to fixed pins on each FPGA, thus constraining the design. In a typical application, additional connections may be required between the FPGAs and/or the daughter board: a number of pins are free on each of the FPGAs and these can be hardwired as required. The need for greater flexibility in interconnection will be one of the issues for the next generation of FPGA card.

An important requirement for any FPGA application is the selection of an appropriate clock rate. If the clock rate is too fast for the complexity of the design, then

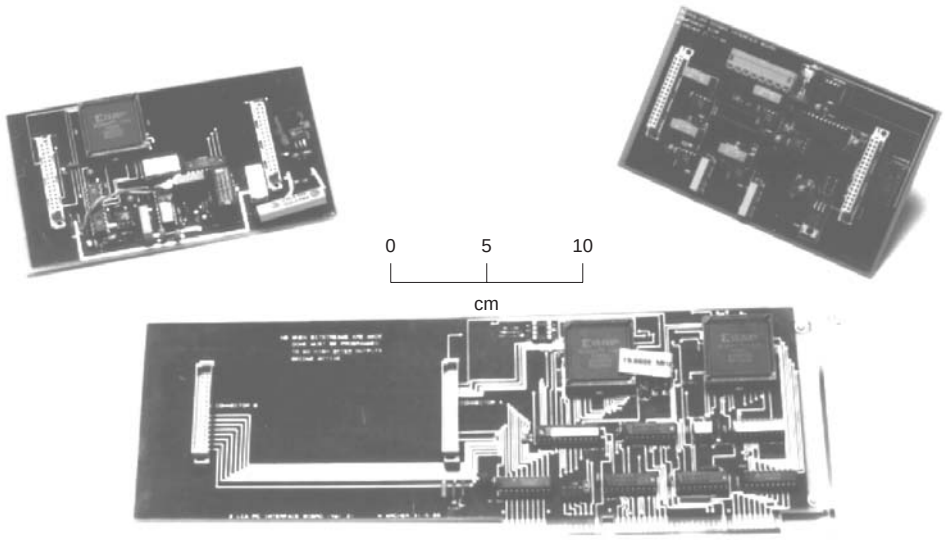

Fig. 4. FPGA card (centre) and daughter boards: (left) thermocouple; (right) dissolved oxygen.

logic propagation from one cycle will not have been completed before the next cycle begins, and errors may result. Provision is made on the card both for low-frequency clocks (using an RC oscillator) such as might be used in, say, stepper motor applications (Henry 1995a), and for high-frequency crystal oscillators in the range 1 - 20 MHz. Both FPGAs share the same clock source to ensure synchronous operation.

3.2. Interfacing to the PC host

Generic array logic chips (GALs) are used to provide the lowest level interfacing to the PC bus, and provide the following facilities:

- The lowest 5 address lines are read from the PC bus, allowing up to 32 locations in PC i/o memory to be used by the card. These locations start at an address defined in a GAL. The use of each location is defined in Handel for the particular application.
- Full 16 bit data transfers are permitted between the internal bus and the PC bus. However, PC access to the internal bus is controlled by the master FPGA to prevent improper reads or writes.
- PC Bus interrupts can be generated by the card.

3.3. Using the card

Using the card is a two-stage process:

- First, the appropriate FPGA configuration must be downloaded from the PC to the card.
- Once the FPGA configuration has been loaded and the application started, there must be facilities for the exchange of data between the PC and the card.

Three locations in PC memory (typically 0x200 - 0x202) are used to reset and configure the FPGAs. Multiple FPGAs are configured in a daisy-chain, including up to three devices on the daughter board. A C program has been written to download configuration files (in the standard .EXO format) from the host PC.

Once an application is running, data are exchanged between the PC and the card via the I/O ports. The number and usage of the ports is defined within the Handel code itself. For example, one port may be written to by the PC software to request active testing or setting changes on the card. Other ports may be read to provide the latest measurement data. In addition, the card has the ability to generate PC interrupts to ensure that new data are dealt with promptly. Note that separate PC programs are typically used to configure the FPGAs and to support the application (e.g., via a graphical user interface).

The card's simplicity ensures that it is easily supported by host software - for example, the thermocouple application has been hosted by PC software written in Visual Basic, C++, and National Instrument's Labview.

3.3. Application 1: thermocouple

The first implementation of a SEVA thermocouple is described by Yang (1993). It used two commercial analogue I/O cards and specialised circuitry to provide raw data to a PC software package. The software carried out measurement and uncertainty calculations, and detection of and compensation for a number of faults.

The current implementation (Atia, *et al.*, 1995) has been used to explore how much SEVA functionality can be included entirely within FPGAs. In practice, a simple but complete thermocouple validation system was implemented using two Xilinx 3195 FPGAs. This system has been used to demonstrate to academics and industrial audiences that simple SEVA devices can be implemented entirely within silicon, for example ASICs.

Figure 5 shows the function performed by each of the major components. The daughter board holds the con-

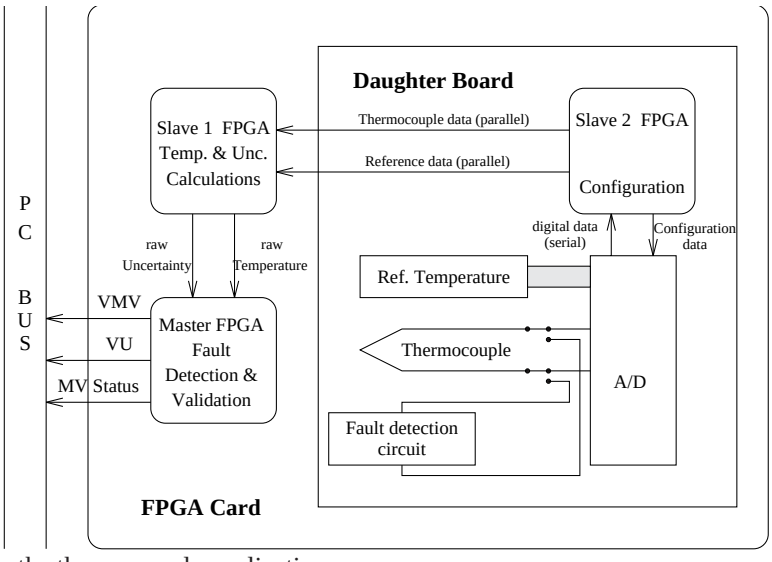

Fig. 5. Hardware for the thermocouple application

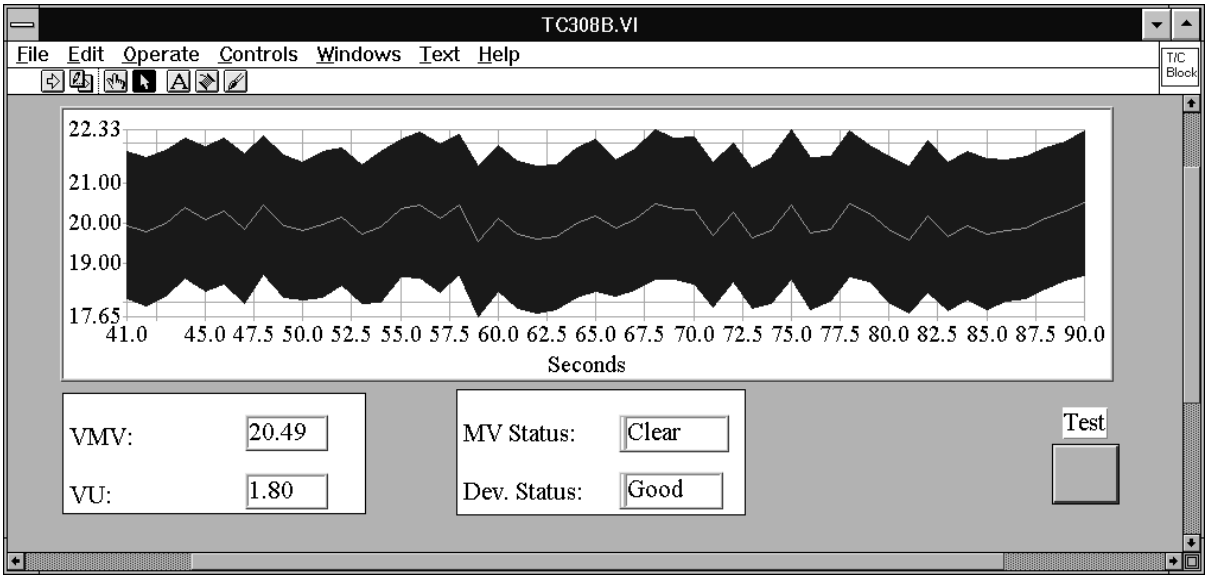

Fig. 6. Labview application interfacing to the FPGA card thermocouple.

nection to the thermocouple, the fault-detection circuitry, the reference temperature circuitry, and a high-precision ΣΔ ADC. All of these components are controlled by an additional FPGA (Slave 2) mounted on the daughter board. The raw data from the thermocouple and reference junction are passed to the FPGA card proper. Here raw temperature and uncertainty are calculated in the first FPGA (Slave 1), using fixed-point arithmetic, while the Master FPGA carries out fault detection, measurement correction, and the generation of the SEVA metrics VMV, VU, and MV status. These parameters are communicated to the host software package via the PC bus.

In this application the minimum requirement of the host software is simply to download the required FPGA configuration and then to collect and display the validated data at regular intervals. Successful implementations have been written in C++, Visual Basic, and Labview. In practice, these different host programs have explored different extensions to SEVA, such as fieldbus (P-net) communication and sensor fusion (Henry, 1995b). For example, Fig. 6 shows the Labview interface to the thermocouple card. Here some additions have been made to the FPGA card functionality described by (Atia, *et al.* 1995): device status is generated, and the user can request active self-testing at any time, in addition to the regular hourly checks.

3.4. Application 2: dissolved oxygen

The second sensor validation project using the FPGA card is a commercial dissolved oxygen sensor (Clarke and Fraher, 1995). In this application, no validation takes place on the FPGA card itself. Instead, the card provides a simple interface to the required set of I/O components for PC-based validation software. The

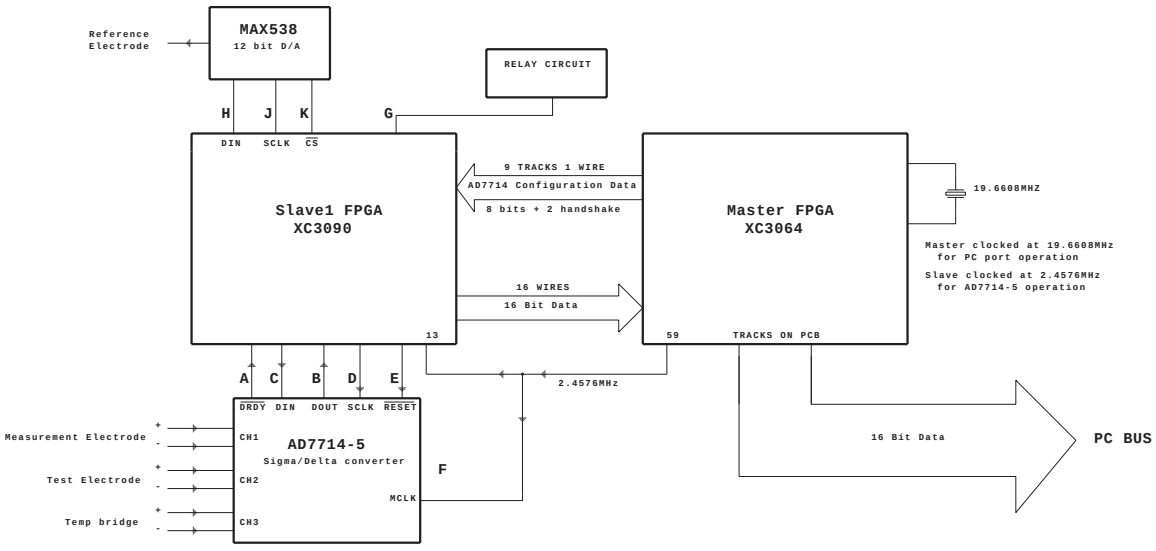

Fig. 7. Hardware for the dissolved oxygen application

commercial transmitter is bypassed entirely, and the daughter board connects directly to the transducer. The interfacing requirements include the following:

- Very high accuracy measurement of voltage from a measurement electrode (to provide uncorrected oxygen data - typically at 20Hz).
- Very high accuracy measurement of voltage from a bridge circuit connected to a thermistor (to provide temperature data).
- Occasional, higher-frequency data from a test electrode (typically at 200Hz).
- A DC voltage applied to the reference electrode.
- The ability to switch between data streams from the test and measurement electrodes.
- Interrupt generation when a new sample is ready.

Figure 7 shows the resulting hardware configuration. The FPGAs provide a set of software ports allowing the PC application control over each of the I/O components:

- One byte-wide port is used for control when written to and provides status data when read.
- Three byte-wide ports are used for reading sampled data and writing configuration data.

The primary daughter board component is the Analogue Devices AD-7714 ΣΔ ADC. It samples from the thermistor bridge, or measurement or reference electrode, as determined by the relay state. At 20Hz, it provides 21 bits of precision. The ADC has complex interfacing requirements, as several on-chip registers control its behaviour. This complexity is hidden from the PC application: the three configuration ports offer reduced options which are adequate for the DOx application. The desired configuration is written to the ports, and then a bit is set in the control port. The FPGA card transmits a complete set of configuration data to the AD-7714. Similarly, when new data is generated by the

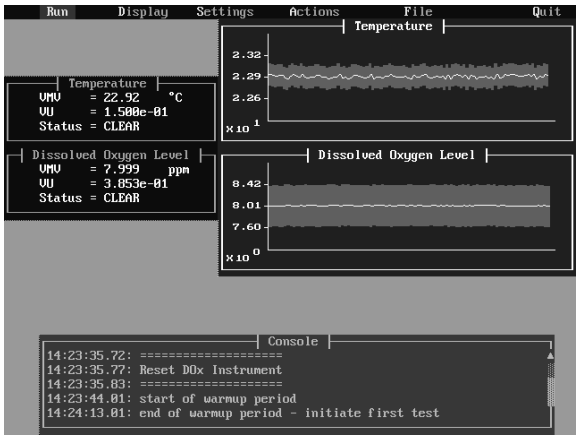

Fig. 9. Dissolved oxygen validation software written in C++ running under DOS.

ADC, either a bit is set in the status byte or an interrupt is generated, and in either case the complete 21-bit sample is ready to be read in the data ports.

A Maxim 538 12-bit DAC is provided to generate output to the reference electrode, and it is controlled in a similar manner. The desired output is written to the lower two data ports, and a control bit is set. Finally, a single bit in the control word sets the relay state.

Figure 8 shows a dissolved oxygen validation program (based on the old Siamese libraries) in operation. Other software packages (e.g., for use with Labview) are currently under development.

4. TRANSPUTER CARD

The transputer card provides a more powerful, flexible, but also more costly, hardware platform for validation. It is based upon the TRAM standard (Inmos, 1988b), which specifies physical, electrical and data requirements for transputer-related modules. The transputer is a micro-processor intended for embedded and parallel-

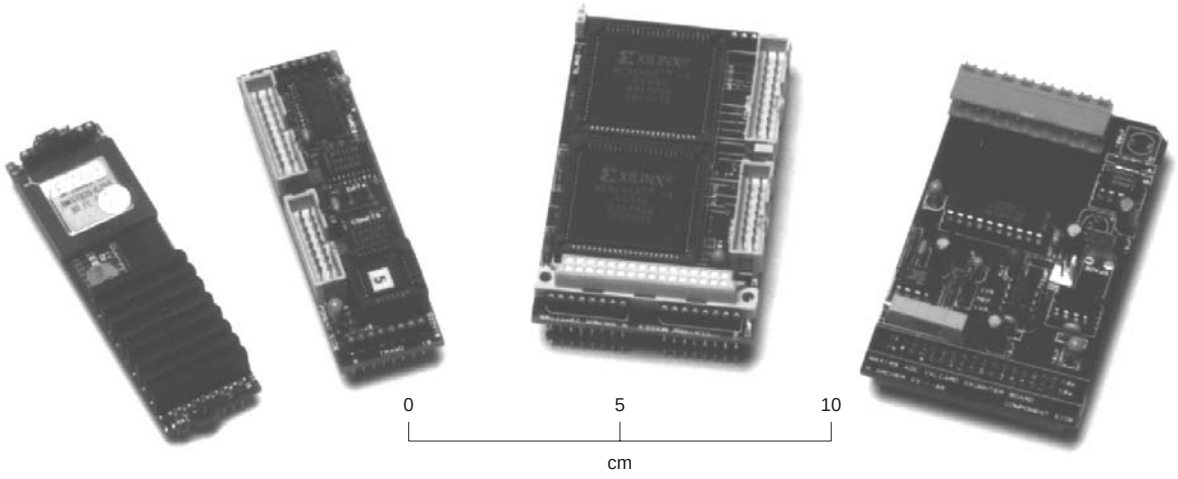

Fig. 8. Transputer card modules (left to right): transputer module, Comtram module, FPGA module, Coriolis-specific daughter board.

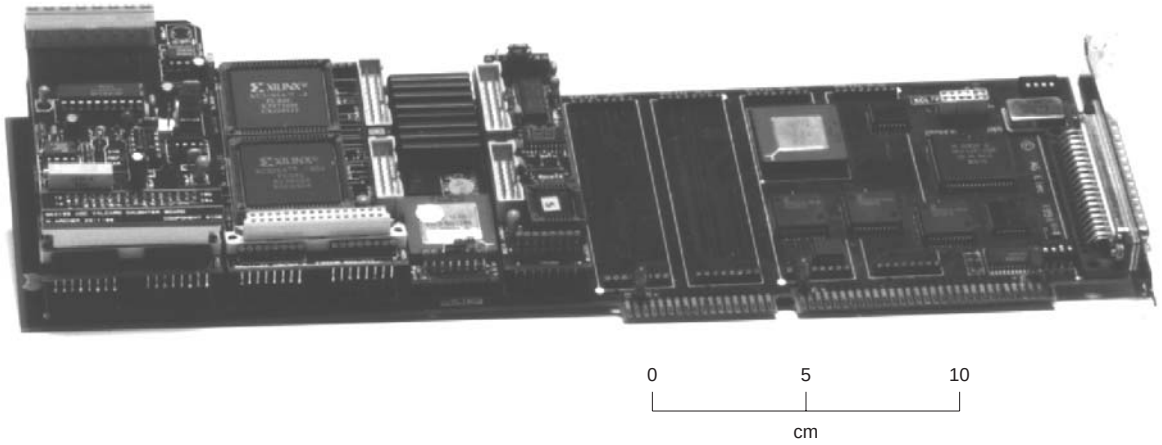

Fig. 10. Transputer card with transputer module and Valcard modules mounted on the host board, excluding bus cabling.

processing applications. Each transputer has up to 4 20MHz serial "links" for data communication. Commercial transputer host boards (an example is shown in Fig. 10) can support several TRAM modules and provide facilities for connecting links between modules as desired by the user. For example, several transputers can be supported by a host board, each processing data in parallel with the others, and communicating via links. Alternatively, a single transputer can control and communicate with modules performing other tasks, e.g., a frame grabber module for image processing, an ethernet link module, or a hard disk module. The transputer has been used as a host for a number of FPGA systems e.g., HARP (Hoare and Page, 1994).

4.1. Nomenclature

Had a single, fixed architecture been developed, it would certainly have been called a "Valcard". Instead a variety of cards and modules were used, some commercial, some developed by the project but generic, and some application-specific. As a consequence, to avoid confusion, consistent nomenclature is an important issue. For clarity, the following terms are defined as they are used in the rest of the paper:

- **Valcard** - a reference to the *principle* of developing flexible hardware exploiting hardware compilation techniques, or to the Oxford-based project. It is *not* the name of any specific piece of hardware.
- **FPGA card** - this refers to the architecture described in Section 3.
- **Transputer card** - this refers to the architecture described in this section, and is shorthand for "transputer-based Valcard architecture". A transputer card consists of a host board, a transputer module, a Comtram module, and one or more FPGA modules, each of which may have a daughter card.
- **Host board** - this refers to a commercial PC card designed to support several TRAM-compliant modules. Any commercial product might be suitable; in this project Transtech's TMB-16 was used (Transtech, 1994).
- **Transputer module** - this is a commercial TRAM module holding a transputer and some RAM.
- **Valcard module** - this denotes either of the TRAM modules developed as part of the Valcard project: the Comtram module or the FPGA module (as described in the next section).
- **Daughter board** - this term refers exclusively to application-specific hardware which plugs into FPGA modules.

4.1. Valcard modules

The transputer card exploits the TRAM standard and link architecture. It is assumed that a single transputer module will be used to control and communicate with all Valcard modules. A link from the transputer module is used to communicate via a "Comtram" module with an arbitrary number of FPGA modules, each of which supports a daughter board holding application-specific circuitry. Figure 9 shows the Comtram and FPGA modules, and a daughter board for the Coriolis application, all of which were designed and constructed at Oxford. Figure 10 shows the various modules mounted on a host board.

4.1. Comtram module

The Comtram module is a size 1 TRAM module, which provides an interface between a link from the transputer module and two distinct buses which are specific to the Valcard architecture:

- The configuration bus is used to configure each of the FPGAs in the current set of modules. The bus is uni-directional and one byte wide (together with control and handshaking lines). Jumpers on the FPGA modules and daughter boards are used to enable the configuration of an arbitrary number of FPGAs via the configuration bus and daughter board connectors: in a series of trials, up to four FPGA modules and one FPGA on a daughter

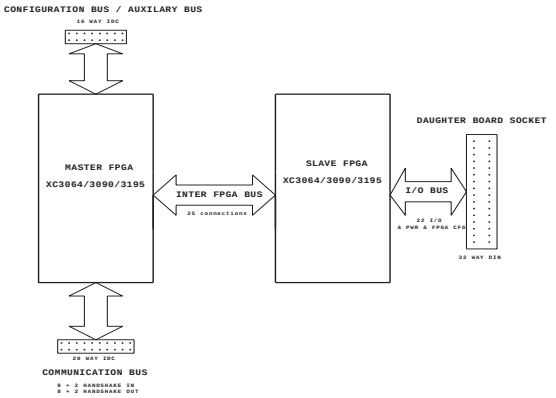

Fig. 11. Schematic of FPGA module

board were successfully configured (i.e., a total of nine FPGAs).

- The communication bus is used to exchange data between the FPGA modules and the transputer once the application software and FPGA configurations have been loaded. The bus is bi-directional, one byte in each direction (plus handshaking). Each FPGA module holds a number of addresses to which the transputer can read or write data via the bus. A key feature is that the exact functionality of the communication bus is defined by the FPGA configurations, and can therefore be application-specific. This aspect of the communication bus is discussed further in the Coriolis example.

In each case bandwidth remains constrained by the 20Mbps serial link, but the use of separate buses allows much greater flexibility than would be possible using conventional transputer link communication between all modules.

On power-up the Comtram module directs all data from the transputer link to the configuration bus. Once the configured line is set (indicating that all FPGAs have been configured), the communication bus is used exclusively for data transfers to the transputer. Various technical difficulties have prevented the use of a single bus for both communication and configuration in the first implementation, although the two buses may be merged in a later design.

4.2. FPGA module

Each FPGA module holds two Xilinx 3000 series devices (Fig. 11). One FPGA (the Master) is connected to the communication and configuration buses. The other (the Slave) interfaces with the daughter board connector, providing the following facilities:

22 general purpose i/o lines

- 5MHz clock signal
- 0 and +5V supplies
- Configuration lines for FPGAs mounted on the daughter board

Distributing the global clock simplifies synchronisation between the FPGA and daughter board components, although a separate clock can be placed on the daughter board if required.

Typically, the Master FPGA deals primarily with bus communications, and the Slave controls and communicates with the daughter board components. If significant data processing takes place within the FPGAs (e.g., in the Coriolis application described later), then higher-capacity devices are used. 25-general purpose interconnect lines are provided between the FPGAs.

4.3. Software support

The software requirements for a transputer-based architecture are rather more complex than those for the FPGA card. The commercial transputer card requires device drivers and software on the host PC if it is to function properly (Transtech, 1994). In particular, in order to execute code on one or more transputers on the card, a DOS program called Iserver must be running on the host PC. Iserver downloads the transputer code and then provides basic services for the transputer(s) such as disk, keyboard and video I/O. Fortunately, source code for Iserver is available, and so its functionality has been extended to provide additional services.

A program called GPUSH (general purpose Seva host) is the result: it supplies all basic transputer services but also provides a Siamese-style user-interface including real-time graphics. Figure 12 shows GPUSH running the Coriolis application. The messages shown on the console indicate the steps taken to set up the application. First the instrument-specific code is downloaded onto the transputer. The first task of the transputer is to reset and configure the FPGAs to provide the required interface to the instrument. Note that the configuration file is on the host file-system, to which GPUSH provides access. Finally, the transputer begins instrument validation, and GPUSH provides a user interface.

At present, host support can only be provided in the DOS environment, using Oxford's Siamese libraries.

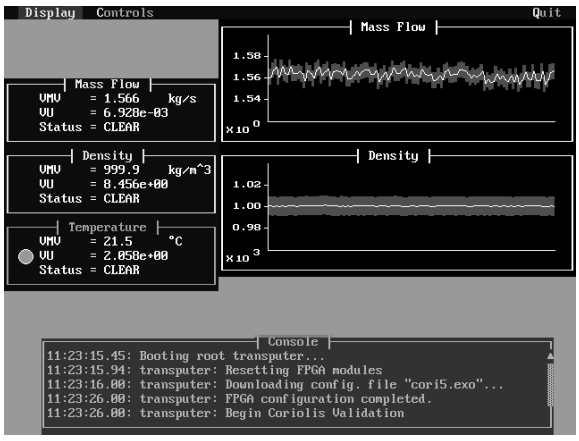

Fig. 12. GPUSH software supporting transputer card

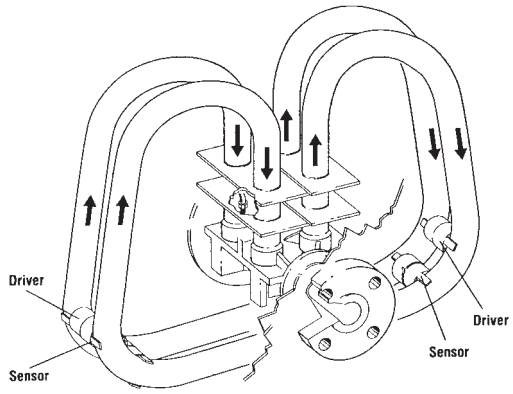

Fig. 13. A commercial Coriolis meter flowtube.

However, Windows device drivers and libraries are under development, so that commercial packages such as Labview can be used to support transputer-based Valcard applications.

5. TRANSPUTER CARD APPLICATION: CORIOLIS MASS FLOW METER

The Coriolis mass flow meter (Henry 1994a) is the most sophisticated instrument for which a self-validating prototype has been implemented. It simultaneously measures the process fluid mass flow, density and temperature by maintaining and observing vibrations in a flowtube (Fig. 13). The flowtube shown operates in counter-phase mode, with the drivers on each side vibrating 180° out of phase with each other. The velocity sensors are used to measure the frequency of oscillation (which indicates the process density), and the phase difference between the two sensors (which is caused by the action of Coriolis forces) is directly proportional to the mass flow of the process fluid.

The Coriolis transmitter contains a microprocessor and a sophisticated analogue control system. For the creation of a self-validating prototype based on a PC, the interfacing requirements are complex: indeed it was the time and expense of creating a suitable interface to the Coriolis meter that partly inspired the Valcard initiative (Henry, 1995a). Table 1 lists the characteristics of the signals which are monitored as part of the validation package.

Table 1. Interfacing requirements for coriolis meter

Signal No.	Signal Type	Frequency Range	Amplitude Range	Input or Output
1, 2	Sine	60-95Hz	1-10V	Input
3, 4	Sine	60-95Hz	10-80mA	Input
5, 6	Sine	60-95Hz	0.1-0.3V	Input
7	DC	-	1-5V	Input
8	Square	60-95Hz	0-5V	Input
9	Square	0.2-1.6kHz	0-5V	Input
10	Square	14-36kHz	0-5V	Input
11	DC	-	0-10V	Output
12	digital	(4 bits)	0-5V	Output

Briefly, signals 1 to 6 are used for diagnostic purposes, and the amplitude of each signal is of primary concern. Signals 7 - 10 are raw measurement data generated by the transmitter electronics - the frequency of signals 8 - 10 must be determined to very high precision in order to retain the measurement accuracy of the meter (e.g., 0.2% of reading for mass flow). Finally, signals 11 and 12 are outputs used to modify the behaviour of the transmitter to improve its performance in the presence of particular fault modes. Note that the sampling rate required by the validation system is 10Hz, so that where possible *average* values should be provided for signals 1-10 for the last 100ms.

Henry (1995a) describes the architecture of the first version of the Coriolis validation system. Here a new implementation is described based upon the valcard-based transputer card architecture.

5.1. Overview of implementation

Figure 14 shows a simple schematic of the Coriolis

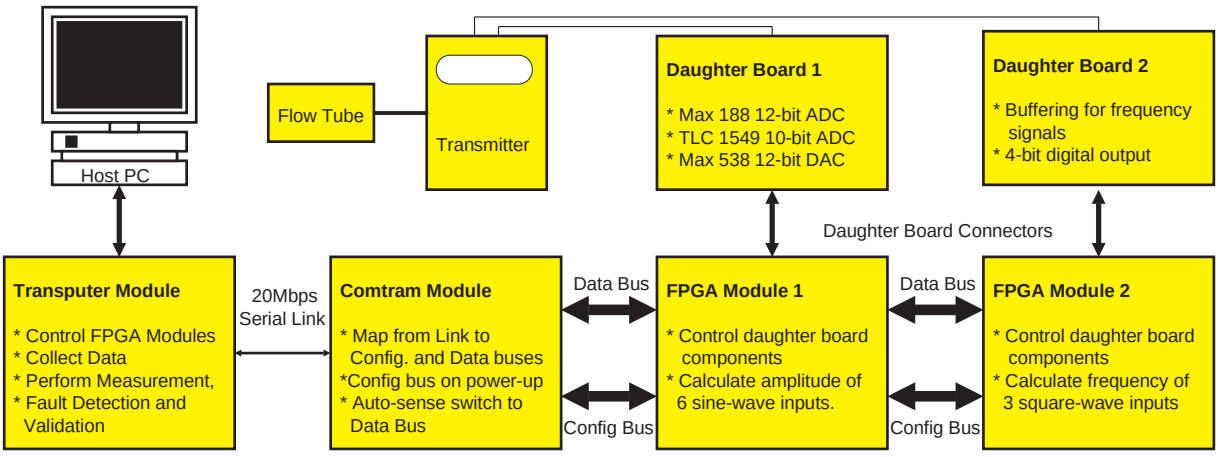

Fig. 14. Transputer card implementation of Coriolis Meter Validation

validation scheme implemented using the Valcard architecture. Each signal pick-off point within the Coriolis transmitter is connected directly to the appropriate daughter board on one of the FPGA modules. Data passes via the FPGA modules to the transputer, which performs measurement, fault detection, and validation calculations, and then passes the validated measurement data to the host PC.

All twelve signals can be dealt with using only two FPGA modules plus daughter boards. In addition, the FPGAs carry out significant signal processing tasks which otherwise would require a substantial increase in the workload of the transputer and/or additional analogue circuitry.

5.2. FPGA Module 1

This module, together with its daughter board, deals with signals 1 - 7. The daughter board is shown in Fig. 9. Signal 7 is a straightforward DC voltage with only a small influence on measurement accuracy and fault detection. It is thus sampled using the Texas TLC 1549 10-bit ADC.

Signals 1 - 6 are all sinusoidal. At any instant they have the same frequency, which lies between 60 and 95Hz, but they have different phases. The property of interest for each signal is its RMS amplitude, which may vary by up to an order of magnitude with changes in operating and fault conditions.

In the original Coriolis validation system, analogue signal conditioning was used to extract an RMS amplitude signal, which was then fed into an ADC. In the transputer card version, very little hardware is used as the conversion is carried out within the FPGA module.

The Maxim 188 is a high-speed 12-bit ADC with built-in multiplexer. Signals 1 - 6 are supplied to the ADC and each is sampled at 10KHz. The FPGA module

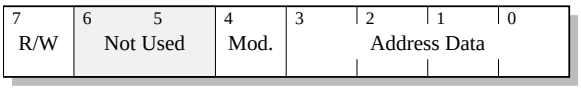

Fig. 15. Structure of Communication Bus Command Byte

rectifies each signal by detecting zero crossings and performing integration using the composite Simpson method. To make the most effective use of FPGA silicon, this calculation is carried out using integer arithmetic; to obtain the RMS amplitude the integral must be scaled by the frequency of the sine-wave, which is carried out by the transputer.

As the frequency of oscillation is typically about 80Hz, while the transputer processes data at 10Hz, the FPGA module measures several waveforms so that an average value over the last 100ms is presented to the transputer.

A further refinement is the use of a Maxim 538 12-bit DAC to generate the reference voltage for the ADC. This ensures that the full ADC range is used as the signals vary in magnitude. Briefly, once the transputer has received data for each channel and calculated the RMS amplitude, a new reference voltage can be assigned to each signal, based on the current amplitude. This reference is used each time the signal is sampled over the next 100ms interval. The transputer must of course scale the resulting data for reference voltage as well as frequency when calculating amplitude.

5.3. FPGA Module 2

This module, and its daughter board, deal with the remaining signals 8 - 12. Signals 11 (a DAC voltage output) and 12 (a 4-bit digital output) are straightforward: the transputer is provided with a port to write the desired outputs, which are then transferred to the corresponding daughter board components.

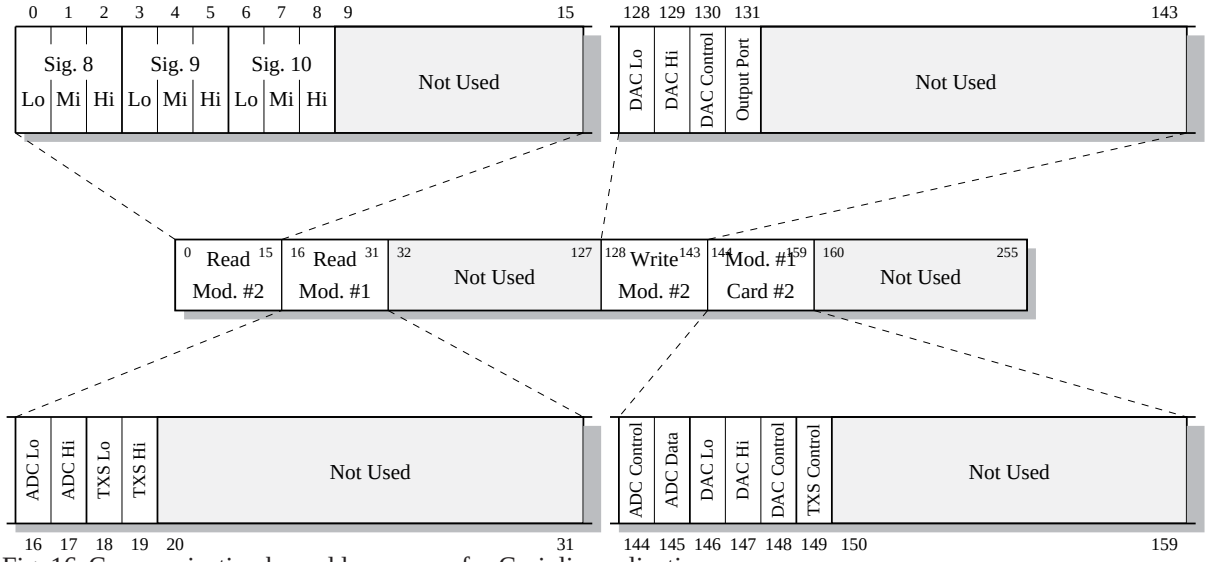

Fig. 16. Communication bus address space for Coriolis application

Signals 8, 9 and 10 provide raw measurement data in the form of square-waves, the frequency of which must be measured to a high accuracy. This is achieved by counting 5MHz clock ticks. As the input frequencies vary from 60Hz to 36kHz (Table 1), appropriate numbers of waveforms are gated while the clock ticks are counted. As a result, the transputer receives average frequency data for the last 100ms.

5.4. Communication bus operation

The communication bus plays an essential role in transferring data between the FPGA modules and the transputer. Bus usage is determined almost entirely by how the Master FPGA on each module responds to the data input lines: in other words bus functionality is defined by the Handel code running within the FPGAs, and can be changed (and indeed subsequently refined and extended) to suit each application.

In the Coriolis example, a very simple protocol has been defined. The transputer commences all bus operations by writing a one-byte command to the communication bus. The structure of the command is shown in Fig. 15.

Bit 7 indicates whether the request is for a read or a write. Bit 4 indicates whether FPGA module 1 or 2 is being addressed, while bits 0-3 provide the address of the byte-wide port of interest within the FPGA module. If the port is to be written to, the transputer sends a byte of data as soon as the command byte is acknowledged. If the port is to be read, the FPGA module sends the required byte of data to the transputer to be read.

The command byte can also be viewed as a single address in a space of read and write ports. Figure 15 illustrates this idea, and shows how ports were actually used in an early version of the Coriolis application. An overview of the address space is shown in the central strip, while the function of each port is shown in the upper strip (FPGA module 2) and lower strip (FPGA module 1). For example, ports 0-2 are read by the transputer to obtain the latest 24-bit estimate of the frequency of signal 8 over the last 100ms. There are plenty of unused locations in the address space, but new ports have been added incrementally to the map as the application has been developed.

This bus protocol provides a simple, if not particularly efficient, interface between the transputer and the FPGA modules. From the perspective of the validation software running on the transputer, requests are sent and data received via a conventional transputer link. A write operation is effected by two byte writes to the link (command + data), while a read operation requires a byte write (command) followed by a byte read (data).

Of course, this functionality is limited, as, for example, there is no method for an FPGA module to actively request a data transfer, nor is data transfer between modules possible, except via the transputer. In addition, single-byte data transfers are not particularly efficient use of bus bandwidth. However, as all functionality is defined in the software running on the transputer and FPGAs, such additional features are readily added. For example, the size of data transfer (say 1,2,4 or 8 bytes) might be included within the command byte using the unused bits 5 and 6 (Fig. 15). Alternatively, reading from or writing to particular ports might by convention entail multiple-byte transfers. This leads to no difficulty as long as both transputer and FPGA module expect the same behaviour. In practice, however, for the Coriolis application, bus bandwidth is not in short supply, and this very simple interface is entirely adequate.

5.5. Transputer and host software

The Coriolis validation software which runs on the transputer is largely the same as the original PC-based software, with some important changes:

- The original program had to provide both instrument and host functionality (e.g., graphical user interface). Most host functionality is now provided by the GPUSH package, which simplifies the transputer code considerably. On the other hand, transputer-host communication has been provided.
- The transputer program includes code to download configuration data to the FPGA modules before validation can commence.
- The data acquisition routines have been modified to operate the communication bus as described in the last section.

The requirements for the PC-based host software, GPUSH, have already been discussed in Section 4.3. This software has served satisfactorily for the Coriolis application. Figure 12 shows the Coriolis validation software running under GPUSH. Additionally, validated Coriolis data are sent (using the Modbus digital communication protocol) to an IA series control workstation for display and monitoring. This is a further demonstration of the viability of the SEVA concepts for use in industrial control systems, and anticipates one of the themes of future work.

6. CONCLUSIONS AND FUTURE WORK

This paper has described two architectures which have been developed to apply hardware compilation techniques to build systems for use in sensor validation applications. In assessing the results that have been achieved, a number of questions may be posed:

- Are the architectures and associated development tools (in particular hardware compilation) suitable for the application domain?

- Is development work more effective using the new architectures?
- Have the new architectures assisted the extension of the research programme into new areas?

Each of these questions will be examined in the following subsections.

6.1. Applicability to sensor validation
As the thermocouple, dissolved oxygen probe and Coriolis meter examples have demonstrated, the architectures developed in the Valcard project have proved to be applicable to a range of sensor validation systems, each with quite different requirements. Correspondingly, the Handel code routines running in FPGAs have carried out a range of tasks:

- The provision of a simple interface for PC software to a set of I/O devices (in the dissolved oxygen application)
- A complete validation task for a simple sensor (thermocouple), including measurement and uncertainty calculation (using fixed-point arithmetic) and diagnostic reasoning.
- Signal conditioning tasks such as sine-wave rectification and high accuracy frequency measurement (Coriolis application)

Given the flexibility shown by these architectures, they may well be adapted to other types of application, and within the Engineering Science Department at Oxford the FPGA card is being offered as a general purpose platform for a wide range of projects.

6.2. Improvements in system development
The experience gained in using the new architectures and tools has justified many of the claims for improving system development work such as creating a new SEVA prototype. The following benefits have been observed in a number of projects:

- The architectures are sufficiently flexible that one or other can form the basis for any SEVA prototype. This implies a high degree of re-use of hardware and host software. Use of PCB manufacturing ensures that cards are available quickly and cheaply for new projects.
- Application-specific hardware can be kept to an absolute minimum, as many tasks traditionally carried out in fixed-function components can be implemented within FPGAs. This can cause a significant reduction in hardware - as demonstrated by the fact that only two FPGA modules and daughter cards are needed for the Coriolis application. Indeed, an entirely separate Coriolis meter could be validated on the same transputer card by the addition of two identical modules.
- The hardware requirements for a project can be settled early and implemented quickly. A variety of FPGA code routines can be written, to test the hardware components in the first instance, followed by incremental development of the application. Many design decisions (such as the division of processing between FPGAs and processor) can be taken late in the project, on the basis of experience using the system. In the case of the transputer card, even the bus protocol can be modified in the light of experience.
- There is a much higher degree of software re-use than in the past. A library of Handel code for driving various hardware components and for common signal-processing techniques is being maintained. The separation of host and instrument software has increased code re-use on each side, and the provision of Valcard links to commercial packages such as Labview reduces or indeed obviates the need for developing and maintaining graphical user interface software.

In the light of these advances, it is intended to use these architectures or their successors in all future development work in the SEVA research project.

6.3. Additional benefits to the SEVA project
A number of new goals have been achieved directly or indirectly as a result of the Valcard project:

- Validation of several instruments within a single host PC. This has been achieved using three thermocouple validation systems, and is being used for research into sensor fusion using SEVA metrics.
- Transmission of SEVA metrics over industrial communication systems, namely the P-Net Fieldbus standard (Henry, 1995b) and Modbus.
- The development of an entire sensor validation application (thermocouple) within hardware, demonstrating that SEVA can be implemented in low-cost technology such as ASICs.

6.4. Limitations of the architectures
A number of criticisms can be made of the architectures. One of the most difficult design decisions was to determine how many pins should be provided for FPGA and daughter board connections. In practice, different applications have different requirements, and fixed interconnections proved insufficiently flexible. In the case of the FPGA card, additional hardwired connections were usually required. This problem can be solved using programmable interconnect devices, which will be used in the next generation of card.

Feedback received from industrial engineers has been positive regarding the rapid prototyping capability, but saw the technology as of limited value for actual industrial use. One concern was the high cost and power consumption of FPGAs. A PC-based card is a useful prototyping tool, but is usually not appropriate as a final product for intelligent instrumentation.

6.5. Future developments

It is intended to develop these architectures. A number of issues have been identified:

- A new generation of FPGA card will include programmable interconnect chips, RAM, and ROM.
- Configurability may be extended to the front-end of the system hardware. through the use of the recently introduced "analog" FPGAs.
- Fieldbus (of one flavour or another) is the future working environment for all intelligent industrial sensors. The provision of a fieldbus capability would allow the development of standalone SEVA prototypes.
- The extension of the rapid-prototyping methodology through to a commercial implementation - for example in the form of an ASIC - could be valuable to the control and instrumentation industry. In this scenario, an instrument design is prototyped in a PC environment such as Valcard, defined in standard languages for both hardware (e.g. Xilinx netlist) and software (e.g. C code), and then mapped directly into a low-cost, low-power ASIC.

7. ACKNOWLEDGEMENTS

This work was partly funded by EPSRC grants GR/J44636 (Valcard) and GR/K48884, and by the Foxboro Company. Thanks are due to Analog Devices for the donation of a range of components.

8. REFERENCES

ANSI (1985). Measurement Uncertainty. ANSI/ASME PTC 19.1-1985.

Archer, N. and J. Bowles (1995). PC Configurable LCA Card. *Internal Report*, Department of Engineering Science, Oxford University.

Atia, M. R. A., J. Bowles, D. W. Clarke, M. P. Henry, I. Page, G. Randall, and J. C-Y. Yang (1995). "A Self-Validating Temperature Sensor Implemented in FPGAs". *Proc. 5th Workshop on Field Programmable Logic and Applications*, eds. W. Moore and W. Luk, Oxford, UK..

Boser, B. E. and B. A. Wooley (1988). The design of sigma-delta modulation analog-to-digital converters. *IEEE J. Solid-State Circuits*, vol SC-23, pp 1298-1308.

Clarke, D. W. (1995), "Sensor, Actuator and Loop Validation", *IEEE J. Control Systems*, August 1995, pp 39-45.

Clarke, D. W. and P. M. Fraher (1995). "Model-Based Validation of a DOx Sensor". *Plenary Paper at the IFAC Workshop On-Line Fault Detection and Monitoring in the Chemical Process Industries*, Newcastle, U.K.

Wilson, P.D., R. S. Spraggs and S. P. Hopkins (1996). " Universal sensor interface chip (USIC): specification and applications outline". *Sensor Review*, Vol 16, No. 1, pp18-21.

Guccione, S. A. and M. J. Gonzalez (1995). "Supercomputing with reconfigurable architectures". *Proc. 5th Workshop on Field Programmable Logic and Applications*, eds. W. Moore and W. Luk, Oxford, UK.

Henry, M. P. and D. W. Clarke (1993). The Self-Validating Sensor: Rationale, Definitions and Examples. *Control Engineering Practice*, 1, 585.

Henry, M. P. (1994). A SEVA Sensor - The Coriolis Mass Flow Meter. *IFAC/IMACS Safeprocess Symposium*, Espoo.

Henry, M. P. (1995a). "Keynote Paper: hardware compilation - a New Technique for Rapid Prototyping of Digital Systems - Applied to sensor validation", *Control Engineering Practice*, Vol 3, No. 7, pp 907-924.

Henry M. P. (1995b). "Fieldbus and sensor validation", *IEE Computing and Control Journal*, Vol 6, No. 6, pp 263-272.

Hoare, C. A. R. and I. Page (1994). Hardware and Software: The Closing Gap. *Transputer Communications*, Willie, 2(2), June 1994, 69-90.

IEEE (1994). VHDL revised standard (VHDL 1076-1993).

Inmos (1988a). **occam2** Reference Manual. Prentice Hall International, Hemel Hempstead, UK.

Inmos (1988b). Transputer Reference Manual. Prentice Hall International, Hemel Hempstead, UK.

Kline, S. J. and F. A. McClintock (1953). Describing uncertainties in single sample experiments. *Mech. Eng.*, pp 3-8.

Page, I. (1994). Parameterised Processor Generation. In *More FPGAs*, W. Moore and W. Luk (eds.), Abingdon EE&CS Books, Oxford.

Page, I. and W. Luk (1991). Compiling **occam** into Field-Programmable Gate Arrays. In *FPGAs*, W. Moore and W. Luk (eds.), Abingdon EE&CS Books, Oxford.

Spivey, M. and I. Page (1993). How to design hardware with Handel. Internal Report, Oxford University Computer Laboratory.

Transtech (1994). Transputer Motherboard User Manual. Transtech Parallel Systems, High Wycombe, UK.

Viewlogic (1991). Workview Manual. Viewlogic Systems, Inc., Malboro Massachusetts, USA.

Yang, J. C-Y. (1993). Self-Validating Sensors. *D.Phil Thesis*, Department of Engineering Science, University of Oxford.

Yang, J. C-Y. and D. W. Clarke (1996). Control Using Self-Validating Sensors. *Trans. Inst. MC*, Vol 18, No 1, pp15-23.

Appendix 1: Handel code to drive the MAX532 DAC from an FPGA

```
(* Handel code for driving MAX532 Dual Channel 12-bit DAC
   Generates a positive ramp on the right channel
         and a negative ramp on the left  channel   *)

use “valslavepins95.hdl”;                                   (* include file giving defaults *)

(* CONSTANTS *)
val datawidth  = 12;                                        (* size of each data register *)
val txregwidth = 24;                                        (* size of transmission register *)
val txshift    =  1;                                        (* shift by a single bit *)
val txbit      = 23;                                        (* which bit to output - top bit *)

(* VARIABLES *) (* length in bits *)  (* tag for simulation *)

val data_left  = Int datawidth       “data_left”;           (* left  channel data *)
val data_right = Int datawidth       “data_right”;          (* right channel data *)
val txregister = Int txregwidth      “txregister”;          (* transmission register *)
val txloop     = Int 5               “txloop”;              (* loop counter *)

(* CHANNELS *)
val txdataChan = Chan 1              “txdataChan”;          (* one bit data stream to MAX532 *)
val sclkChan   = Chan 1              “sclkChan”;            (* clock signal to  MAX532 *)
val csChan     = Chan 1              “csChan”;              (* data ready signal to MAX532 *)

(* Channel Protocol Converters  - interface definition for each channel *)
val txdataChanCPC  = (CPC_NhPortOut [ IO_21 ] , Output txdataChan);
val sclkChanCPC    = (CPC_NhPortOut [ IO_20 ] , Output sclkChan);
val csChanCPC      = (CPC_NhPortOut [ IO_19 ] , Output csChan);

prog := Handel
(
    [ txdataChanCPC, sclkChanCPC, csChanCPC ],              (* CPCs used *)
    [ data_left, data_right, txregister, txloop ],          (* Variables used *)

Seq [                                                       (* Start of program code *)

    Par [                                                   (* Initialisation *)
        data_left  := C 0,
        data_right := C 0,
        txregister := C 0,
        txloop     := C 0
        ],

    While (TRUE,                                            (* Main loop - endless *)
        Seq[
            While ( txloop != C txregwidth,                 (* for each bit *)
                Seq[
                    Par [                                   (* set up data and rising edge *)
                        sclkChan   !! C 0,                  (* output low on clock line *)
                        txdataChan !! txregister BIT txbit  (* output data bit *)
                        ],
                    Par[                                    (* generate rising edge *)
                        sclkChan   !! C 1,                  (* output high on clock line *)
                        txdataChan !! txregister BIT txbit, (* output data bit *)
                        txloop     := txloop + C 1,         (* increment loop *)
                        txregister := txregister < txshift  (* shift transmission register *)
                        ]
                    ]
                ),

            Par [                                           (* data transfer complete *)
                csChan     !! C 1,                          (* flag data complete to DAC *)
                txloop     := C 0,                          (* reset loop counter *)
                data_left  := data_left  - C 1,             (* negative ramp *)
                data_right := data_right + C 1              (* positive ramp *)
                ],

            txregister := data_left ^ data_right            (* assign new data *)
            ]                                               (* end write/read Seq *)
        )                                                   (* end While *)
    ]                                                       (* end program Seq *)
);                                                          (* end program *)
```