A Systems Re-engineering Case Study: Programming Robots with occam and Handel-C

Dan SLIPPER and Alistair A. McEWAN

Applied Formal Methods, Department of Engineering
University of Leicester, University Road, Leicester, LE1 7RH

Abstract. This paper introduces a case study exploring some of the legacy issues that may be faced when redeveloping a system. The case study is a robotics system programmed in occam and Handel-C, allowing us to draw comparisons between software and hardware implementations in terms of program architecture, ease of program code verification, and differences in the behaviour of the robot. The two languages used have been selected because of their model of concurrency and their relation to CSP. The case study contributes evidence that re-implementing a system from an abstract model may present implementation specific issues despite maintaining the same underlying program control structure. The paper identifies these problems and suggests a number of steps that could be taken to help mitigate some of the issues.

Keywords. LEGO Mindstorms NXT, FPGA, software architecture

Introduction

The application of systems thinking to a systems development process (as described by Hitchins in [1]) encourages the use of abstract modelling. This allows us to ensure the behaviour of a system meets its specification and can be verified before implementation. Using a mathematical model to verify software systems during design is a common process. This is done at a high level of abstraction resulting in a Platform Independent Model (PIM). A PIM can then be verified before implementation and is beneficial in early development stages, putting emphasis on the functional correctness of the model. Beyond this an implementation can be targeted to either hardware or software, avoiding dependencies on specific platforms during the design phase.

A PIM of a system is useful if problems occur, such a component failing and the original component being unavailable for replacement. In this situation being able to redevelop from a PIM will be easier and resultantly cost less to rectify. In industrial applications this reduction in cost is important. Smith et al [2] emphasise the need for correctness in the early development stages and how this reduces risks caused by change later in life. It is common within the automotive or military industries that a product can have a required life of decades—in which time some components will degrade and therefore such systems will eventually require maintenance or replacement. Riffle presents an example in [3] of aircraft component failures resulting in unavailability and how a lack of platform independence in the design can introduce challenges in fulfilling the initial product specification.

"Through life information management" is an important part of system development and significantly aids the re-engineering process. Utilising an existing PIM from the original development (rather than extracting the model from an existing system, as highlighted in [4]) can aid re-engineering and result in a more accurate basis from which to work. The

work presented in this paper studies some of the issues present when a legacy system is re-developed from an existing PIM on to different hardware or with a different implementation language.

The re-engineering process is of particular interest in cases where completely different hardware platforms are used. In the case of an aircraft flight control system, if a controller board were replaced during its life then redevelopment could be to a processor based system or to reconfigurable hardware. A redevelopment would require verification that it meets the original specification. In such safety critical systems the tolerances of these systems are crucial and our paper demonstrates this by the use of a simple case study.

Our case study for re-engineering has been focused around an embedded system and designed to utilise two implementation languages inspired by CSP: occam and Handel-C. These languages have different implementation targets. occam was originally designed for the Transputer but can also be executed on a more generic processor via the Transterpreter virtual machine [5]. Handel-C is a Hardware Description Language (HDL) for development on a Field Programmable Gate Array (FPGA) which augments the semantic style of ANSI C with the concurrency model provided by CSP. The target platform in question is the LEGO Mindstorms NXT kit [6], an educational robotics platform on which control code can be programmed and interfaced with various sensors and motors. The Transterpreter has been ported to this platform enabling execution of occam on the processor. The case study within this paper describes a system based around the sensors and motors of the NXT, re-engineered to use Handel-C for implementation onto a FPGA. This new implementation is then tested against the original occam system, using control code (developed following the same code structure). The contributions of this paper are:

1. Introduce a port of the Transterpreter for the LEGO Mindstorms NXT.
2. Demonstrate a common program structure between hardware and software architectures (using Handel-C and occam).
3. Highlight the integration issues faced whilst re-engineering the system.
4. Discuss the benefit of using CSP based languages from system design and development perspectives.

The context of these goals are introduced in section 1, with a background understanding of the technologies used in the experiment and motivation for the work. Section 2 details the implementation specific considerations for development of the two systems. Section 3 introduces a case study developed for both platforms and section 4 presents the final test results. Finally section 5 draws some conclusions from the work, and introduces the future direction of the research.

1. Background

This section presents a background to the NXT robotics kit, the languages used within the experiment described in section 3, and the hardware on which they are implemented.

1.1. LEGO Mindstorms Hardware

The Lego Mindstorms NXT robotics kit is predominantly designed for use as an education tool. The kit ships with the graphical programming language NXT-G, which enables users to program robots via drag and drop boxes (which represent sensors and functionality). LEGO now have two types of NXT kit, NXT and the more recent NXT 2.0. For this case study the standard kit is used, this consists of a number of sensors; this includes an ultrasonic sensor capable of measuring the distance from objects, a sound level sensor, a light sensor and a touch sensor. It also contains 3 servo motors, each of which has a built in tachometer for

precise control. Internally the NXT brick is similar to many embedded platforms, with its main processor being an ARM7TDMI. Alongside this it also has an Atmel AVR co-processor handling interactions to the sensors. A novel peripheral of the NXT hardware over its predecessor (the RCX) is its Bluetooth controller, able to maintain a list of known devices and allowing communication with up to 3 slave devices or a single master device.

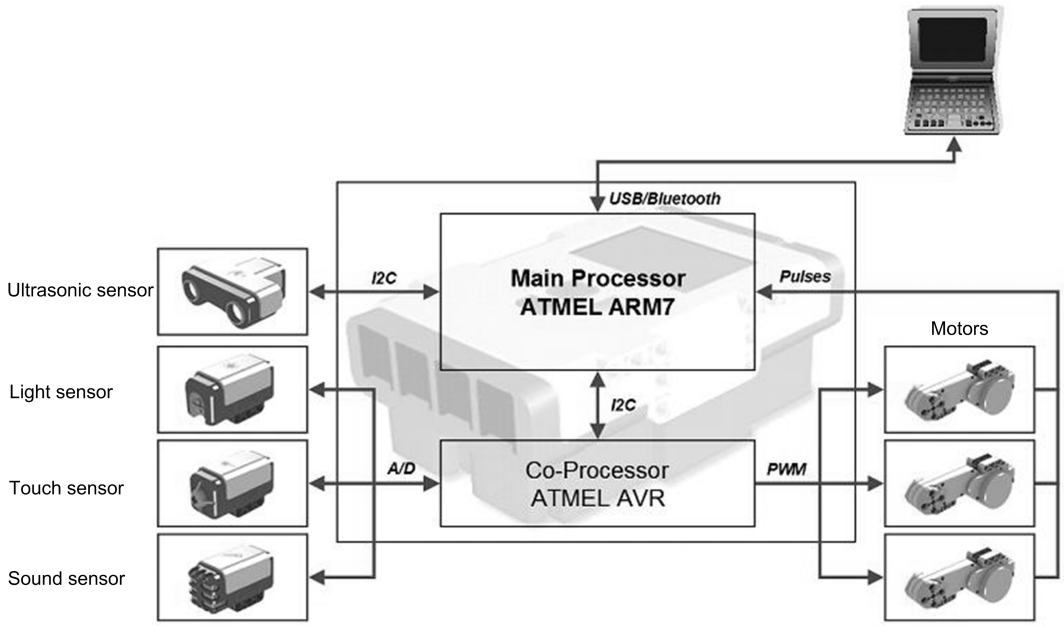

Figure 1. The NXT Hardware Architecture.

The hardware within the NXT is shown in Figure 1. This also details the protocols used for communication with each of its peripherals. A number of languages have been developed for use on this platform including C [7] and Java [8]. Other languages have been developed to support concurrent programs, such as [9].

1.2. Modelling and Concurrency

Some software systems are developed to have a number of different processes of control executing in parallel. In such systems, the communication between these components can become difficult to manage. This complication can lead to a series of problems within the code such as deadlock, livelock, and race conditions.

A number of methods of overcoming these issues and modelling such systems have been proposed. Petri nets [10] are of particular use for describing distributed applications, and modelling how the control flows within a program. Other models of concurrency can also be used, such as Calculus for Communicating Systems [11] and Algebra of Communicating Processes [12]. The language of particular interest in this paper is Hoare's Communicating Sequential Processes (CSP) [13]. The notion of *refinement* in CSP permits abstract specifications to be designed and reasoned about (using model checking tools such as FDR2 [14] and Probe[15]), before being translated to an implementation language.

1.3. CSP-Inspired Programming Languages

CSP has inspired the development of a number of programming languages, permitting the development of abstract specifications, and then following a set of transitions to translate the specification into an executable program. The first of these languages, occam [16] was developed for use with the INMOS Transputer, an early microprocessor designed specifically

for parallel execution, as introduced in [17]. The Transterpreter virtual machine allows for occam code to be interpreted on any processor that can execute ANSI-C.

Other software implementation languages have been developed specifically from the model of concurrency introduced by CSP, such as the recent development of Google's Go [18] and Erlang [19]. A number of others have spun off as libraries providing CSP functionality, for instance PyCSP [20] and JCSP [21]. Further work has been undertaken beyond JCSP to produce JCSPre [9], a reduced version of the environment specifically for the LEGO NXT.

CSP has also inspired HDLs such as the library expansion upon the Verilog language, VerilogCSP. This library provides CSP style communication on top of the standard language. However there are languages which are written specifically to follow the CSP model. Handel-C is one such language [22], and has been described as "literally occam with a C like syntax" [23]. Unlike Verilog and VHDL, the style of programming is for more representative of writing standard software than using a HDL. The occam and Handel-C languages have been selected for this case study, due to their close relation to each other, CSP, and ease of programming for a software engineer. This also allows us to build on previous work in deriving programs from a CSP specification [24].

1.4. occam and the Transterpreter

The Transterpreter is a virtual machine for interpreting occam-pi bytecode on different architectures. Developed as part of the Kent Retargetable occam Compiler (KRoC) project, the Transterpreter is a small C library ported to operate on a number of embedded platforms, including the predecessor of the Lego NXT (the RCX platform) as described in [25].

The Transterpreter also offers the capability to develop functionality (in our project, device drivers) using the underlying C language and call it from occam. Issues may occur if a C function does not return correctly and can cause unexpected side effects to the behaviour of the occam code, reducing the reliability and predictability of the system.

1.5. Handel-C and FPGAs

FPGAs are made up of a number of configurable logic blocks, each of these can be connected together via I/O blocks and routing channels. Synthesis of an FPGA configures the routing between each of these blocks, combining the blocks together to provide functionality. Development boards (such as the one used in this experiment) also have a number of external peripherals that can be utilised. In comparison to developing an Application-Specific Integrated Circuit (ASIC), FPGA technology has various benefits. The reconfigurable nature of FPGAs removes the need for a full re-manufacturing of a circuit at any point in the development cycle unlike an ASIC—where changes to a design may require a completely new manufacturing process. Evidence of this difference in design times has been shown in a white paper by Xilinx [26], presenting a comparison between ASIC and FPGA development cycles. The results demonstrated that an FPGA development can be up to 55% shorter due to the instant nature of redesign and testing.

Other benefits include the ability to configure an FPGA to emulate the behaviour of a processor as described in [27]. This can allow for co-design of processors and bespoke hardware within the FPGA itself. This is a powerful tool for combining the advantages of the two different methods into a single system. In the case study presented in this paper, a Spartan 3A development board was used. The particular FPGA chip onboard is the XC3S700A [28], with 700k gates and over 400K RAM bits (either block or distributed).

2. Testbed Design

In this section we present the software design on the NXT platform and how this has been re-engineered to execute in hardware on the FPGA. The software model has been created around a PIM, providing the same processes and communications on both systems.

2.1. Software Process Structure

Developing a control algorithm for a robotics based system can follow a variety of paradigms. [29] reviews a number of different methods and how they can be utilised while programming control with occam-pi. This paper highlighted that for a case study such as ours the control algorithm should be as simple as possible, allowing the use of basic robotic control theories as introduced by Brooks in [30]. This paper presented a methodology for layering processes for the input, output and control separately. This structure and the flow of processes and communications between them is depicted in Figure 2. Each of the processes contains an infinite loop which continues execution, (or blocking) until a command has been received or until the program receives a STOP condition—at which point the system will cease execution.

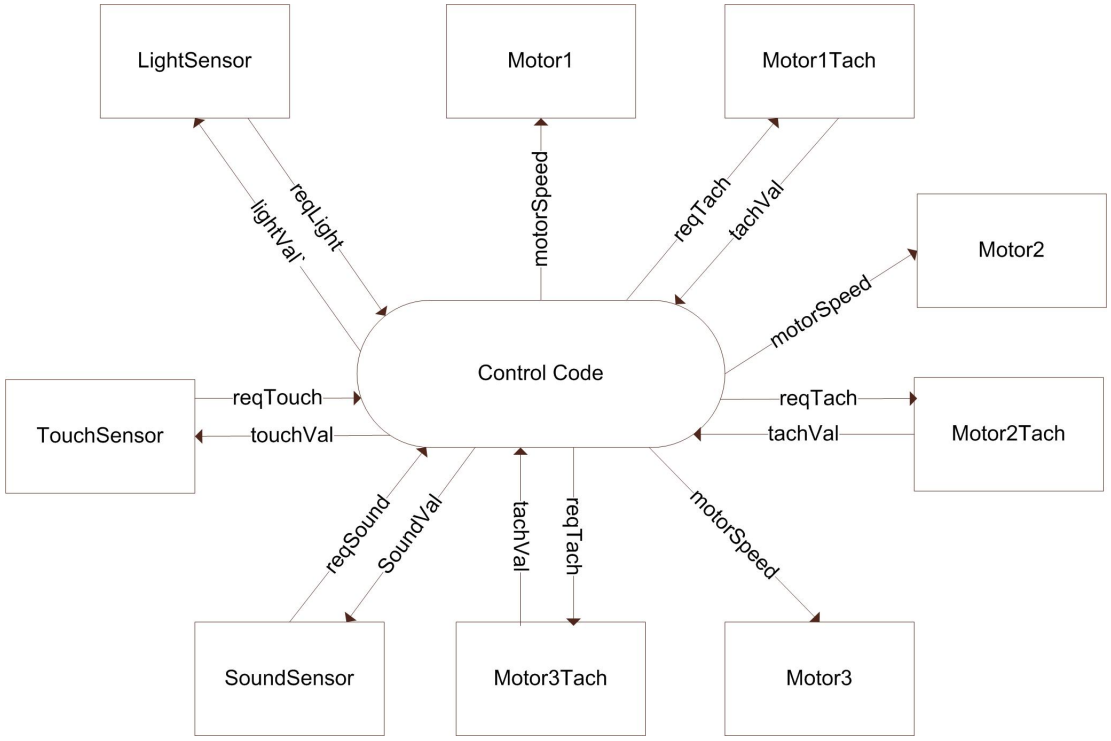

Figure 2. The NXT process structure.

2.2. NXT Architecture

The system re-engineered for this paper is the Transterpreter port to the NXT. This virtual machine allows occam to be executed on the platform, whilst utilising an existing third party operating system layer to communicate with drivers for the hardware and sensors. The NXOS operating system is an open source development, required to boot the ARM and AVR co-processor. This base OS provided C driver libraries to interface with the sensors, motors, and other peripherals. Utilising this functionality from the operating system layer removes the need for development of device drivers purely in occam. Figure 3 shows how the different layers of the NXT software system tie together.

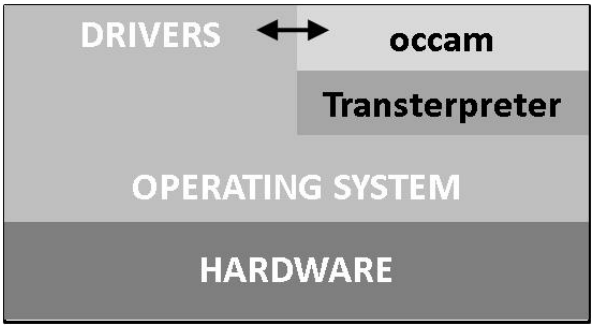

Figure 3. Layers of the NXT software base.

This architecture allowed for occam wrapper processes to be created around the libraries, in the process structure previously described, using channel communications to request and send data between processes. The developments of JCSPre and the initial development of the Transterpreter for the LEGO RCX [25] both demonstrated a similar structure to the one undertaken for the NXT.

2.3. FPGA Implementation

Re-engineering of the NXT system onto an FPGA requires drivers for the sensor and motors. As previously shown in Figure 1, the system uses a number of protocols—in an industrial project, data like this should be documented and available for reference throughout the systems life. The NXT hardware guide details the pin outs from the brick to each of the sensors, allowing a similar wiring setup for the FPGA. Figure 4 shows the FPGA with connection points for the motors and sensors highlighted.

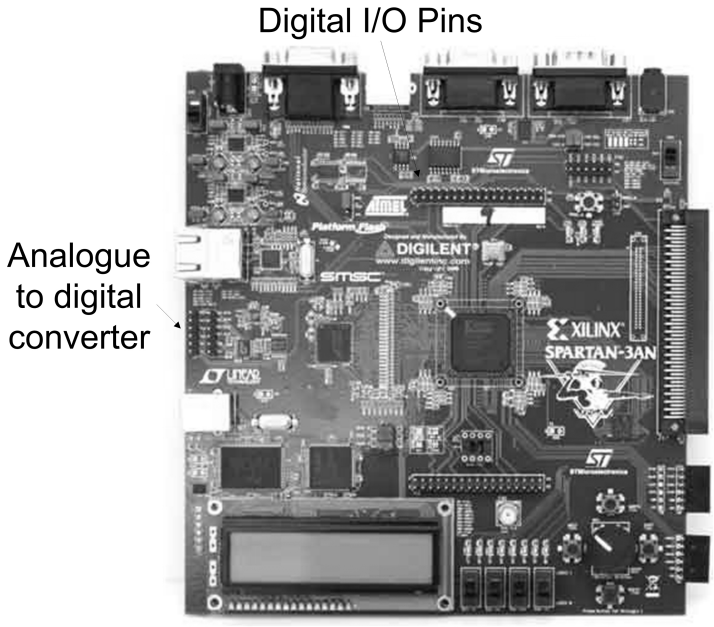

Figure 4. Spartan 3A development board.

3. Case Study

After the structure and interfaces for the system were redesigned for the FPGA implementation, the two systems were programmed with a common control process designed to utilise

the sensors and motors in a number of ways. This case study was to simply follow a pre-programmed path around a desk, making turns with a number of different speeds, directions, angles and delays. The light sensor was also utilised by recording the light level along the desk and recognising when it reached the edges. Using a case study like this allowed for multiple sensors and motors to be used in parallel, whilst avoiding the introduction of additional control code complexity. This way the experiment would provide clear visual results of the differences in control.

3.1. Design of Experiment

To remove all possible variables which could introduce inaccuracies during the case study the following actions were devised to remove or reduce any possible impact on the results:

- Battery Life - to ensure different levels of charge between experiments did not introduce differences in speeds and angles of turning, batteries were fully charged before each test.
- Size of the robot - the NXT and FPGA have a very significant size difference, to counteract the effects of this, the two controllers were combined into the same robot frame. Since the two carried around both the NXT and FPGA, the weights were identical for each experiment.
- Method of turning (by time/angle) - the tachometer incorporated into the motors was used allowing the robots to turn a given angle.
- Test area lighting - to recognise when the robot has reached the end of the desk it used a light sensor. The experiment was conducted in dark conditions, removing possible interference from sunlight. Both systems were calibrated to recognise the same level as light or dark before the experiment was carried out.

3.2. Control Code

The main process structure from both implementations is presented in this section, and discussed further as to how they can affect the behaviour of the two systems. Full source code has been included in an online repository[1].

```
1  void main (void){
2      chan int 1 fromTouch, fromLight, reqTouch, reqLight, reqTachA,
           reqTachB;
3      chan int 32 toMotorA, toMotorB, motorATach, motorBTach;
4      par{
5          ADC_Read(fromTouch, fromLight);
6          LightSensor(fromLight, reqLight);
7          TouchSensor(fromTouch, reqTouch);
8          Motor1(toMotorA);
9          Motor2(toMotorB);
10         ControlCode(fromTouch, fromLight, toMotorA, toMotorB,
11             reqTouch, reqLight, reqTachA, reqTachB, motorATach,
                   motorBTach);
12         Motor1Tach(motorATach, reqTachA);
13         Motor2Tach(motorBTach, reqTachB);
14     }
15 }
```

Listing 1. Handel-C main process

[1]Code can be found at https://github.com/ds151/Re-engineering-paper

```
1  PROC main ()
2    CHAN INT motorASpeed, motorBSpeed, fromTouch, fromLight, reqLight,
         reqTouch, fromTachA, reqTachA, fromTachB, reqTachB:
3      PAR
4        sensorLightA(fromLight, reqLight)
5        sensorTouchB(fromTouch, reqTouch)
6        motorA(motorASpeed)
7        motorB(motorBSpeed)
8        controlCode(motorASpeed, motorBSpeed, fromLight, fromTouch,
             reqLight, reqTouch, fromTachA, reqTachA, fromTachB, reqTachB)
9        tachCountA(fromTachA, reqTachA)
10       tachCountB(fromTachB, reqTachB)
11 :
```

Listing 2. occam main process

Listings 1 and 2 show the main process for both systems. Note that despite the language and the platform they are implemented on being different, the underlying process and channel communication structure is consistent throughout. The main difference that can be highlighted in these code samples are how the Handel-C version of the code assigns widths for channels (see lines 2 and 3 of Listing 1). This is due to the nature of redeveloping to hardware. The data from the touch sensor for example is known to be a single bit, and so unnecessary data lines are not built into the circuit design. Another difference in the main code is that the FPGA version utilises an extra function to read the ADC and update the appropriate variables (as shown on line 5 of Listing 1). Within the occam system, these calls are hidden inside the driver code.

Contained within the online repository is the control code that was used for the experiment, showing how the two systems were tested with the same control algorithm. An overview of this code is that initially the variables utilised are set up, followed by the first stages of the experimental path that the robot will follow. The first stage is the two motors being activated in parallel to drive to the end of the desk, at this point the motors will stay on until an appropriate light level is seen. In the occam version this is a value straight from the ADC, however in the case of the FPGA this value is hard coded into the driver as a request for a light or dark level. Beyond this the motors are activated based on timings and angles, producing a number of turns around the programmed path.

4. Results

The previous section introduced our case study and some samples of the code from the systems. The overall program control code is the same and the drivers beneath were developed to follow the same structure, despite differences in the ADC resolution and sample rates of the tachometers. The only significant differences were in the syntax and variables used (for example, integer width as previously discussed). During testing the Handel-C implementation did not display the correct behaviour at one point around the fixed path it was following and needed to be stopped as it would be unable to stay on the desk for the rest of the experiment (representative of tolerances in a system making its operation unsafe). This result was found to be the same for every execution of the Handel-C system.

Further tests were carried out to examine the cause of these problems since they should not be related to the control code. A simple test was devised to determine the behaviour of a small part of the code. This was to drive a single motor until the tachometer count reached 1000 pulses and then stop the motor. The way in which the motors are controlled within this test is significant. Both the occam and Handel-C systems used “float” method of stopping the motors. This entails cutting power to the motor, at which point it rotates until it loses

momentum. The alternative method would have been to engage the motors briefly in a reverse motion locking them in the current position. Results were gathered of the tachometer counts from the occam and Handel-C implementations once the robot had come to a stop, based on tests at 2 different speeds. Each test was repeated 5 times.

Table 1. Results of test to stop the motor at 1000 pulses, travelling at full speed.

Robot	Run 1	Run 2	Run 3	Run 4	Run 5	Mean	Range	Standard Deviation
Handel-C	1247	1240	1239	1233	1232	1238	15	6.058052
occam	1773	1783	1793	1814	1787	1790	41	15.26434

Table 1 shows the results of 5 runs with the motors on full speed until the pulse count reached 1000, and some analysis of these values. The values from these executions show that the overrun in pulses was much higher in the occam system and with a much higher range. To clarify these results, the same was run again with a lower motor speed. This test was executed with the motors at 80% speed. This involved a PWM mark-space ratio of 80:20 giving the motors a lower voltage and hence a lower speed.

Table 2. Results of test to stop the motor at 1000 pulses, travelling at 80% speed.

Robot	Run 1	Run 2	Run 3	Run 4	Run 5	Mean	Range	Standard Deviation
Handel-C	1213	1208	1209	1203	1206	1208	10	3.701351
occam	1397	1417	1431	1415	1413	1415	34	12.1161

The results in Table 2 show a lower number of pulses once the robot had stopped. This is to be expected as the motor did not have as much momentum. The previous two tests were carried out with the motor off the table and with no load. To confirm how much difference the physical pressure from the robots weight made on this control, the robots were also programmed to travel the same number of pulses at the same speed. This gave some interesting results of terms of accuracy.

Table 3. Distance travelled after 1000 tachs, with motors at 100% speed.

Robot	Run 1	Run 2	Run 3	Run 4	Run 5	Mean	Range	Standard Deviation
Handel-C	73.8	71.3	68.6	69.6	68.6	70.38	5.2	2.207261
occam	70.4	70.4	71.0	70.6	71.2	70.72	0.80	0.363318

The occam implementation appears more consistent from the results in Table 3. This is as expected as it was the original system. The range and standard deviation in these runs show that occam is very accurate when it comes to stopping the motors whilst under physical pressure from the robot. The interesting statistic within this table is that the mean distance travelled from both the platforms were very close, the range of values from the Handel-C implementation however showed a fair amount of inconsistency in the re-engineered system. In this experiment we expected the system to repeat the same behaviour every time, although some margin of error should be accounted to the fact that robots cannot have precise control unless they have a level of feedback. Martin presents this in [31], even describing how the motors of the NXT are an example of how a robot can have feedback from their actions throughout execution. This said, the test results shown in Table 3 show a much higher margin of error in the Handel-C implementation than expected. This raises issues of reliability in the new system in comparison to the original, and proposes questions as to what could be changed within the development process to avoid such issues in future projects.

Further experiments identified a difference in motor voltage from the two systems. The power to the motors was lower on the Handel-C implementation than the occam system. This is expected to be due to the components used (as the NXT uses a small IC, whereas

the FPGA used an external driver board with larger components). This also explains some of the differences shown in Tables 1 and 2, and why the overrun in pulses is much lower in the Handel-C system.

5. Conclusions and Further Work

This paper presents a case study exemplifying issues present in re-engineering an embedded control system, utilising the Transterpreter as a basis to execute occam code. To do this, a port of the Transterpreter has been introduced for the Lego Mindstorms NXT platform.

This platform was used as the basis for re-engineering to FPGA, using the Handel-C language, structuring the code in this development identically to the occam implementation. Despite using languages that have the same model of concurrency, the results have shown that it is not easy to recreate a system with the exact behaviour of the original. Although there has been significant research within the area to demonstrate that CSP can be used to specify and verify behaviour of software that has been written in occam and Handel-C. The behavioural differences are expected to be due to the base implementations, and the major contributing factor has been identified as a hardware component on which the system was developed.

The results from a re-engineering perspective show that the process followed within this experiment has not provided a fully sufficient system to replace the original in a safe manner. Future redevelopments would benefit from following a number of steps during the analysis and design of new systems:

1. The extraction or retrieval of a platform independent model.
2. Components with verifiable behaviour, i.e. it is clear when the components are functioning as expected. This way it is possible to have 100% certainty that drivers are working and implemented in the correct manner.
3. Physical margin of error – It is possible that a re-engineered system will have slightly different tolerances between components, timing or behaviour. In this situation, the original specification for the application should be consulted to verify that these are within reasonable bounds.

Further work from this study could involve producing a CSP model of the control and driver code for both system and using model checking tools to verify their behaviour formally. This way the correctness of the Handel-C component of the re-engineered system can be proven, and demonstrate problems that are occurring at a hardware level.

Acknowledgements

This research was sponsored by EPSRC and AWE. We would also like to thank Carl Ritson and Jon Simpson of the University of Kent for their work on the Transterpreter NXT port.

References

[1] D. K. Hitchins. *Advanced Systems Thinking, Engineering, and Management*. Wiley, 2003.

[2] M. R. Emes, A. Smith, and A. James. Left-shift vs the time value of money: Unravelling the business case for systems engineering. In *INCOSE Spring Conference*, 2007.

[3] T. Riffle. Reverse engineering & re-engineering of avionics legacy components. *The 21st Digital Avionics Systems Conference Proceedings*, 2:12C3–1–12C3–9, 2002.

[4] F. C. D. Young and J. A. Houston. Formal verification and legacy redesign. *Proceedings of the IEEE 1998 National Aerospace and Electronic Conference*, pages 627–638, July 1998.

[5] C. L. Jacobsen and M. 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, volume 62 of *Concurrent Systems Engineering*, pages 99–106, Amsterdam, The Netherlands, 2004. WoTUG, IOS Press.

[6] LEGO. The LEGO Mindstorms NXT. `http://mindstorms.lego.com/en-us/whatisnxt/default.aspx/`, March 2011.

[7] T. Chikamasa. nxtOSEK Project. `http://lejos-osek.sourceforge.net/`, 2011.

[8] leJOS. leJOS Project. `http://lejos.sourceforge.net/`, 2011.

[9] A. Panayotopoulos J. M. Kerridge and P. Lismore. JCSPre: the robot edition to control LEGO NXT robots. In Peter H. Welch, Susan Stepney, Fiona A.C. Polack, Frederick R.M. Barnes, Alistair A. McEwan, Gardner S. Stiles, Jan F. Broenink, and Adam T. Sampson, editors, *Communicating Process Architectures 2008*, volume 66 of *Concurrent Systems Engineering*, pages 255–270, Amsterdam, The Netherlands, 2008. WoTUG, IOS Press.

[10] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, 1981.

[11] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[12] J. A. Bergstra and J. W. Klop. Algebra of communicating processes. *Mathematics in Computer Science*, 1986.

[13] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[14] Formal Systems Inc. FDR2 user manual, 2005.

[15] M. Leuschel and M. Fontaine. Probing the depths of CSP-M: A new FDR-compliant validation tool. In *Proceedings of the 10th International Conference on Formal Methods and Software Engineering*, ICFEM '08, pages 278–297, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] INMOS Corp. *Occam Programming Manual*. Prentice Hall Trade, 1984.

[17] A. Kent and J. G. Williams. The Transputer family of products and their applications in building a high-performance computer. *Encyclopedia of Computer Science and Technology*, August 1998.

[18] Google. The Go programming language. `http://golang.org/`, 2009.

[19] J. Armstrong. *Making reliable distributed systems in the presence of software errors*. PhD thesis, Royal Institute of Technology, Stockholm, 2003.

[20] J. M. Bjørndalen, B. Vinter, and O. Anshus. PyCSP – communicating sequential processes for Python. In *Communicating Process Architectures*, pages 229–248, July 2007.

[21] Peter H. Welch. Process Oriented Design for Java: Concurrency for All. In P.M.A. Sloot, C.J.K. Tan, J.J. Dongarra, and A.G. Hoekstra, editors, *Computational Science - ICCS 2002*, volume 2330 of *Lecture Notes in Computer Science*, pages 687–687. Springer-Verlag, April 2002.

[22] I. Page. Closing the gap between hardware and software: hardware-software cosynthesis at Oxford. In *Hardware-Software Cosynthesis for Reconfigurable Systems*, 1996.

[23] R. Sharp. *Higher-Level Hardware Synthesis*. SpringerVerlag, 2004.

[24] A. A McEwan. *Concurrent program development*. PhD thesis, University of Oxford, 2006.

[25] J. Simpson, C. Jacobsen, and M. C. Jadud. A native Transterpreter for the LEGO Mindstorms RCX. In *Communicating Process Architectures 2007*, Concurrent Systems Engineering. IOS Press, 2007.

[26] Karen Parnell and Roger Bryner. Comparing and contrasting FPGA and microprocessor system design and development. Technical report, Xilinx, July 2004.

[27] R. Cayssials and E. Ferro. An uRT51 real-time processor evaluation. In *Proceedings of the 14th IEEE International Conference on Emerging Technologies & Factory Automation*. IEEE Press, 2009.

[28] Spartan-3A Starter Kit. `http://www.xilinx.com/products/devkits/HW-SPAR3A-SK-UNI-G.htm/`, 2011.

[29] J. Simpson and C. G. Ritson. Toward Process Architectures for Behavioural Robotics. In Peter H. Welch, Herman W. Roebbers, Jan F. Broenink, Frederick R.M. Barnes, Carl G. Ritson, Adam T. Sampson, Gardiner S. Stiles, and Brian Vinter, editors, *Communicating Process Architectures 2009*, volume 67 of *Concurrent Systems Engineering*, pages 375–386, Amsterdam, The Netherlands, 2009. WoTUG, IOS Press.

[30] Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[31] F. Martin. Real robots don't drive straight. *Proceedings of the AAAI Spring Symposium on Robots and Robot Venues: Resources for AI Education*, March 2007.