Hardware Objects of the circuits for Robotics

Masatoshi Sekine  Takashi Kanamaru  Kenji Kudoh  Haruki Imanaka

Yusuke Shiga  Hikaru Ito  Yoshihiro Myokan

Tokyo University of Agriculture and Technology
www.sekine-lab.ei.tuat.ac.jp

Abstract

A hardware object is a capsule of circuits loaded into reconfigurable devices. It is dynamically constructed while an application runs. Its driver is temporarily mounted on a real driver for a hardware module. The circuits are suitable to pre-processing and post-processing of robot systems. A commercial C++ compiler becomes an integrated design environment only attaching hardware object components and hardware object libraries.

1 Introduction

Reconfigurable devices as FPGAs and PLDs have been much developed to design commercial commodities just in time[3],[6]. There have been several efforts to create object-oriented hardware models for the reconfigurable systems. In an early effort on DECPerle-I by Vuillemin and his group [4], a C++ based tool with the inheritance is used to describe programmable active memories (PAM). They have surveyed leading-edge computational challenges: a RSA cryptography, a long integer arithmetic, a molecular biology, a heat and Laplace equations, a neural network, a high-energy physics, an image acquisition, a multi-standard video compression, a servo vision, a sound synthesis, and a long viterbi decoder. He has summarized that PAM technology had proved superior, in both performance and cost, to all that days forms of general purpose processing systems.

However, the traditional hardware objects are designed for getting high-performance in computation, and also in cooperation with real circuits in ASIC chips[8]. Then, their relative slowness puts off them from the mainstream of devices in cooperation with ASIC chips. In the field of robotics, reconfiguration speed compared with the ASIC chips is not the most significant subject, but easiness in design is most desirable. To utilize these reconfigurable devices, the object-oriented programming method is suitable to make them be flexible hardware components like software objects, because almost sophisticated robot components are controled by software. Davis [2] has proposed a hardware object model derived from a Java thread. It runs on a commercial ACEcard with ACEruntime. He supposed three things that need to be done to create an application with the hardware objects: 1) The hardware objects required by the algorithm must be created either in-house or purchased. 2) Algorithms using those hardware objects need to be developed. 3) The available reconfigurable devices must be managed. The runtime makes the loading, executing and swapping of hardware objects possible.

Bellows and Hutchings [1] reported on a JHDL language and HWS system that includes the JHDL simulation kernel and runtime support classes. They interested in run-time reconfiguration, and they proposed the partial reconfiguration approach through an additional interface class, named for the Partially Reconfigurable Socket. This class is used to describe parts of the circuit that require partial configuration, by using Java, C++ or C languages.

All of the proposed hardware objects are loaded and released by their constructor and their destructors in a common manner. This paper proposes one approach to design style or implementation of software applications that consist of hardware and software objects. Secondly, novel object managing system is proposed to handle the hardware objects directly from software without minute knowledges of them. Our hardware object model (hwObject) has exploited many of these efforts, mainly much dependent on the Davis’s hardware object[9].

However, in contrast to their works, the primary objective of hwObject is different from the previous ones. The hwObject is considered as a primary component working with software objects in a software-dominated application which has a lot of real time functions. We have designed a hardware object of the wavelet transform for example. Another purpose is to set up a base of design methodology of a hardware and software heterogenous system that is originally same as a robot system. The main part of the system is a hwObject manager and interface protocol between the hardware object and software objects.

2 hwObject-oriented application in robotics

2.1 Circuits for complicated situation

The circuit for robots will be something of parallel computation as an circuit of an image processing, a watchdog operations for various sensors, or controls for many servo motors. Among of them, the image processing must proceed a large quantity of information instantly, and preprocessing is necessary in order to solve this demand. So a coarse to fine template matching by wavelet transform will be necessary as the preprocessing as shown Figure 1.

The other hand, mobile machine such as robot have to move under the complicated situation. Accordingly it must be loaded various functions in a circuit, and the most suitable function must be selected according to the complicated situation. Reconfigurable circuit solves such a problem, because various circuits are stored in
compact bit-stream data in a memory device. Novel reconfigurable devices are rapidly developed in order to fast circuit reconfiguration in a few nano-seconds. The other side, commercial FPGA devices which are cheap, large scale and high performance, become easily available to match with demands required by the almost robot systems. So far, we suppose that frequent and high-speed reconfiguration may not become a serious problem for the circuit for robotics.

1.2 Growing up and learning

It is a purpose of learning to acquire a new function or to do well than before. For the circuits of the robotics, the circuit should acquire a new function at the next stage when the circuit has got less efficiency with primary functions. The following items must be considered in case of circuit growing up. 1) Circuit scale is magnified for getting a complicated function step by step. 2) A circuit is adjusted if it went wrong. 3) Memory contents are updated to get more sophisticated information. 4) A function is replaced, or be added with a better one. In order to get the above facilities, the circuit device must be reconfigurable, and also circuit data must be refineable by learning algorithm. For example, the coarse eye template shown in Figure 1 - (d) will get new relations to more finer eye templates during a grow-up phase.

2.3 Multi-layer system

A hwObject-oriented application(hwApp) for robotics is considered to be a multi-layer system, as shown Figure 2, that is commonly found in a peripheral nervous system and such a central nervous system as a brain. Each lower peripheral subsystem makes finer operations autonomously under the more coarse controls of the upper subsystems. The finer information is so digested as to generate more coarse and abstract information at each sensor subsystem.

On the contrary, the more coarse instructions are decoded to the finer instructions and signals by adding finer local information at each motor subsystem. These subsystems seems to be more suitable, if made in circuits rater than in software programs. The hwObject is designed for both subsystems. The other hand, all the decisions are made to generate coarse instructions with the abstract information at the central nervous system. These decision making is more suitable for software programs, because deep condition nesting is necessary for global and multilateral complicated cases. However, there are closely bindings between the central and peripheral nervous systems, the hwObjects must be integrated in the application program, indeed, in the inner codes of the condition nesting.

2.4 Application structure

The hwApp comprises of objects of software and hardware objects, and it is executable after installing hardware modules (hwModules) on a computer with a host processor. The hwApp should be read tens of sensor inputs in time for driving much peripheral apparatus simultaneously. All the functions acting on local data have to be designed to run in the hwObjects if it is reasonable to run locally in parallel, in pipelining, or in distributed processing. On the other hand, central functions dependent on global variables should be executes on the host processor as a necessity. Therefore, the design of the hwApp begins with specifying all the functions to be global or local, central-processing, pre-processing or post-processing.

The host processor has to perform the central processing functions over global data in the main memory, but other functions can be moved to the hwObjects, more precisely, to the hardware circuits(hwNets) with local memories in the hwModule. Then, it is necessary to design data structure for avoiding narrow bus bottle-
neat. That is to say, a quantity of data transmission to the host is reduced by data processing locally done in the hwModule as much as possible. For example, a wavelet transform is utilized as the pre-processing function for coarse-to-fine template matching in a visual recognition system. The central function acts only on the coarse image data, and it acts on the part of the original fine images in the specified domain at last. The other hand, a translation of a superior instruction of the central function is extended to a lot of lower/primitive control instructions/signals by the post-processing. Also, searching functions or watchdog functions will be local functions that perform in parallel over locally distributed data such as lists or trees.

2.5 Temporal hwObject allocation

Some hwObjects have been allocated from the beginning of the hwApp, but many hwObjects are temporally allocated during the execution time of the hwApp. Therefore, a hardware device driver has to be not only attached on the hwApp when the hwObject is constructed, but also it has to be detached in case of the hwObject destruction. Our experiment tells us that a hwNet loader circuit allocated a 200k gates circuit in around 40ms in case of xilinx spartan II-XC2S200. The allocation time of higher family devices is very short as a matter of course. In general, a situation of the application, especially of a robot program, is changing rapidly and it is unexpected at the beginning. Therefore, hardware drivers of the hwObjects have to be attached or detached in time under various conditions by the hwApp itself.

2.6 Design system

All the hwApps will be derived from a hwObject-oriented design system that supplies not only design tools during design and verification phases, but also run-time components for execution phases of the hwApps. An experimental base system, named Mandara, is written in Borland C++ builder as shown in Figure 3. A design and information management of hwApp are done with Mandara DesignForm. Mandara has a special TopModule for purpose of having hardware information of the computer, on which the hwApp is loaded, and hwModules information. Also, Mandara has another object, named "host" that has a hwObject manager(objectMgr/hwMgr) and one virtual object "UserObjectForm". The hwApp, shown as "my_hwObject_App", is derived from UserObjectForm. The hwApp has usually a main state machine for managing hwObjects. Furthermore, Mandara consists of primitive objects for hwObject design such as nodes, edges, nets, event handler, net-editing functions of node-add/delete/move/bind, access facilities of hwObject libraries and design database, objectMgr, etc. Thus, Mandara is easily obtainable by attaching the hwObject library to any (commercial or non commercial) C++ compiler.

2.7 Design flow

Figure 4 shows the design flow of the hwApp. The function level verification schemes are identically equal to ones on a C++ programming using a source code debugger. The total system test proceeds by simply replacing the behavior models of the hwNets with the real hwNets. The hwNets are supposed to be designed in HDL/RTL models manually using commercial EDA tools and tested thoroughly in order to put them on the hwObject library. In addition, their design level is easily extendable up to the high-level synthesis steps, and the HDL/RTL models will be generated from the behavior models written in C++/C, SpecC, or systemC. Standard hwObject classes are supplied by a hwObject library analogous to a standard library or a visual component library. Any user hwObjects are derived from the base classes of hwObjects in the similar way of inheritance that is matured in the object-oriented programming. Then, designers can verify their system level designs with their behavior models of the hwObjects written in the object-oriented software language.

The object manager is capable of an event driven simulation, and the last application is seamlessly ob-
3 Hardware Object and Hardware Module

3.1 System overview

Figure 5 - (a) shows our reconfigurable system, which is structured in an ordinary architecture consisting of one host processor, a main memory (MM), Hard disks, a PCI-bus, and the hwModules simply attached on the PCI-bus[9]. The hwObjects share the heavy load of the host processor with each other, and a host processor can concentrate its generic computing power on more functions that have complexity indeed. In other words, the hwObjects do not replace the ASIC chips, but they work among software objects and the hardware components. From the view of the hwApp, the hwObjects are constructed and destructed with their data area in the external memory domain extended from the main memory area. Therefore, all the hwObject accesses are simple move operations between both areas.

Of course, this ordinary architecture of the reconfigurable system is chosen for its simplicity, especially to use a conventional operating systems. To avoid the bus bottleneck of the PCI-bus, multiple buses or more complex and high performance buses must be exploited. However hwObject accesses remain in simple move operations.

3.2 hwModule

Our hwModule is a memory device that is attached on the PCI bus of a computer (Figure 5 - (b) ). The hwModule consists of one FPGA (Xilinx XC2S200) for PCI and local bus (LB) bridge with a hwNet controller (PCI/LB-IF), 3 local 1MB memories (LM), 3FPGA’s (Xilinx XC2S200) for the hwNets, a 32bit local processor (LP), and general purpose interface ports (GPIF). The hwModule is integrated as a memory-like device, because of its easy accesses from the functions of the hwApp. When an operating system (OS) wakes up, it attaches a device driver for each hwModule but no driver of a hwNet.

Figure 6 shows the configuration in the FPGA. Each FPGA has two address ports, two data ports, one GPIF port, PCI/LB-IF controlling ports, LP controlling ports, and address/data ports connected to the LP. Each FPGA has a peripheral circuit for controlling the hwNets. Each FPGA can be reconfigurable while other ones are running independently. The reconfiguring time is around 40ms for 200K gates by the hwNet loader circuit in the PCI/LB-IF, with 160KB bitstream data from the LM. LP snoops always the hwNet status for making reports to the objectMgr.

3.3 PCI/LB-IF bridge

PCI/LB-IF is a PCI interface with LB control facility shown in Figure 6. The LB control is simple hand-shake operation. The 32bit-wide LB is divided into three buses for purpose of pipelining or parallel operations, each of which can connect with each other by a peripheral circuit inside the FPGA under the control of PCI/LB-IF. All the hwNets have to request to get a grant of LB bus access to the PCI/LB-IF. As a last facility of PCI/LB-IF, the hwNet loading circuit is included.

3.4 hwObject

The hwObject base class is partitioned into four parts as follow:

- Interface buffer and IO ports with the hwNet (buffer, localBuffer) and software objects allocated in the main memory.
4.2 Object manager and hwNet device driver

The objectMgr is mounted on an hwApp at the beginning and it makes a reconfigurable hwModule images by reading hwModule configurations from the topModule. Then the objectMgr manages hwObject and software objects, and it administers the control information in hwNet device drivers, such as LM addresses and buffer sizes, timing data to link the hwObject interfaces, commands and status definition, and a hwNets name. The hwNet device driver is mounted temporarily on the hwModule device driver at the hardware manager (hwMrgr) level of the objectMgr. Thus, the OS does not recognize the hwNet in any case though the hwNet is a hardware. API functions programmed in the objectMgr catch window message events thrown by the window manager of the OS or other objects, and it translates them into events for putting on the priority queue if needed. The events activate hwMgr to write/read data to/from the LM. The objectMgr sends the commands to special area where the LP watches all the time. Then the LP controls the hwNet through the interface in the FPGA as shown in Figure 6.

4.3 Separation of hwNet device driver from the hwModule device driver

As the hwModule is a PCI-device, the OS finds it in the PnP procedure by which the hardware device driver is mounted stationary on the OS kernel. The hardware device driver has the administrative information for the real device attached on the PCI-bus physically. The hwNet is just a circuit, i.e., a real hardware loaded temporarily, thus the hwNet device driver is must also mounted temporarily. However the hwModule is attached on the PCI-bus continuously, the hwModule device driver has to be mounted as long as the hwModule is attached on the PCI-bus. This is a primary reason of separation the hwModule device driver from the hwNet device driver.

5 hwObject Example

5.1 Template matching

Our object localization algorithm works by localizing a matching domain at the finer resolution level by a matching result with an input image and the template at the more coarse resolution level. It shows many steps in the preprocess suitable for the hwObject. One function of the steps is an Haar wavelet transform. For a domain selected with $D^i$, it is used for the control of matching by adding energy of a circuit. A color image (Lenna example in Figure 1) is used as input pattern. For each RGB componentst is 8 bits, and wavelet transform is done 4 times for every RGB component, and wavelet coefficient $W^j(R, G, B)$ of the RGB component ($j=3$ to $0$) (including a scaling coefficient) is saved. Also, the

<table>
<thead>
<tr>
<th>Function</th>
<th>Gate counts</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wavelet Transform</td>
<td>20Kgate</td>
<td>54MHz</td>
</tr>
<tr>
<td>Template matching</td>
<td>18Kgate</td>
<td>43MHz</td>
</tr>
<tr>
<td>Target tracing</td>
<td>28Kgate</td>
<td>42MHz</td>
</tr>
</tbody>
</table>

Table 1: Circuits for hwNets

- Member variables/functions in a MM and a LM.
- Reconfigurable circuits (hwNets) assigned by hwNet name.
- Temporarily loadable hwNet device drivers that includes physical information of hwNet (gate counts, timing, interrupt tables, driver routines, and so on).

The constructor of the hwObject base class registers its instance to the objectMgr. The hwNet accesses data only on the LM. The write/read member functions simply transfer the data between MM and LM. Our hwObject model gets benefits of the simplicity in programming by using the direct controls or accesses to the hwNet.

3.6 hwObject Library

The hwObject uses no details of the hwModule, but it has only port snaps to connect logically with the hwNet. Also, the hwNet is designed by Hardware Design Language so as to be resynthesizable, and the local bus protocol and hwNet operation are simple. Thus, the hwObject and the hwNet are so simple that it is expected to make open free libraries, i.e., free circuits for robotics, as same as free GNU softwares. You can make a free circuit for robotics easily with free softwares and free circuits, i.e., free hwObjects.

4 Event programs

4.1 Event handler

The hwApp has two event handlers, because both time scales of hwObjects and software objects are different. Logic gate primitives such as AND, OR, NOT, etc., are also prepared for a logic simulation. The structure level net description is translated into a net using such primitives as node objects with pin objects and edge objects. One of the event handlers acts as a simulation kernel for verifying a behavior model of the hwNet, and the other is used for the objectMgr. The simulation kernel watches events thrown by the logic gates as the hwObject, and the other watches events occurred in the hwNets. Both of them interrupts the other by sending the events. Asynchronous events are issued when the hwNets send end signals or interrupt signals through the hwModule device driver and the hwNet device drivers. The control codes in the hwApp are written using while(), if(), go(), wait() or break(), and one can manage the hwNets directly through the both event handlers and the events.
eye template \( O^j \) (shown in Figure 1) is made manually selected from the input image, and wavelet transform is done. Before template, matching, intensity \( Y \), and chrominances \( C_r \), \( C_b \), are calculated as following equations,

\[
\begin{align*}
Y &= (11 * R + 59 * G + 30 * B)/100, \\
C_r &= R - Y, \\
C_b &= G - Y.
\end{align*}
\]

Then, their deferences of the template \( O^j \) and the input image \( I^j(x^i) \), \( dY \), \( dC_r \), \( dC_b \), are calculated, and a total defference \( \Delta \)is obtained by adjusting with a mixing coefficient \( K_e \),

\[
\Delta(x) = dY + K_e * (dC_r + dC_b) \quad (2)
\]

Now, the matching value is obtained as a sum of deferences between \( \text{weight}(x) \) of each pixel of the template and \( \Delta \)

\[
D^j = \sum_x |\text{weight}(x) - \Delta(x)| \quad (3)
\]

After the preprocess, the localization is done by selecting the part of the input image \( I^j(x^i) \) having minimum matching value. Thus, it is preferable that many preprocess steps for the template matching would be replaced by the hwObject operations. For the first step, we have developed the hwNet of the Haar wavelet transform that compressed the input image in the hwNet directly transferred from COMS image senser C3188A.

### 5.2 Target tracing

Next hwNet example is a target tracing with one processing element that uses an image compressed with wavelet transform for a template matching[5]. Testing it using a template image of 32*32 pixels(level 1) to an input image of 128*128 pixels(level 1), a matching time is obtained as 221.7ms. Because a picture element number decreases by 1/4.1/16 when input images and template images of level 2, level 3 of the wavelet transform are used, each matching time becomes 55.4ms or 13.8ms. Accordingly it may be said that a realtime chase is possible because level 3 is not more than 30ms. If eight PE is arrayed as other methods, matching time becomes not more than 30ms.

### 6 Summary and Conclusions

The hwObject model is proposed to be suitable for designing the application in robotics. It is noticeable that the requirements for hwObjects working among the software objects are different from ones for executing among the ASIC chips. It seems that the circuit types may be different. We think the circuits for Robotics is suitable to the hwObject with the reconfigurable devices rather than conventional ASICs. The hwModule is designed to explore the feasibility of the reconfigurable devices to the object-oriented system design. The hwObject is necessarily divided into the hwNet in the FPGA and software interface parts performed by the host processor. The granularity of the hwNet is roughly estimated from the circuit for the visual recognition. The manager programs show that both of the hwModule device driver and the hwNet device driver are necessary for performing the hwApps. The hwObject model design environment is explained to be easily obtainable by attaching the hwObject library to any (commercial or non commercial) C++ compiler with any operation system.

### References


