ReConfigME: A Detailed Implementation of an Operating System for Reconfigurable Computing

Grant Wigley, David Kearney and Mark Jasiunas [1]
Reconfigurable Computing Laboratory (RCL)
Advanced Computing Research Centre
University of South Australia
Mawson Lakes SA 5092 Australia
Grant.Wigley@unisa.edu.au David.Kearney@unisa.edu.au Mark.Jasiunas@unisa.edu.au

Abstract

Reconfigurable computing applications have traditionally had the exclusive use of the field programmable gate array, primarily because the logic densities of the available devices have been relatively similar in size compared to the application. But with the modern FPGA expanding beyond 10 million system gates, and through the use of dynamic reconfiguration, it has become feasible for several applications to share a single high density device. However, developing applications that share a device is difficult as the current design flow assumes the exclusive use of the FPGA resources. As a consequence, the designer must ensure that resources have been allocated for all possible combinations of loaded applications at design time. If the sequence of application loading and unloading is not known in advance, all resource allocation cannot be performed at design time because the availability of resources changes dynamically. In this paper we present an implementation of an operating system that has the ability to share its FPGA resources dynamically among multiple executing applications.

1. Introduction

With the development of reconfigurable computers containing FPGAs with in excess of 6 million system-gates, such as the RC2000 [1] and Bioler 3 [2], it is now feasible to consider the possibility of sharing the FPGA between multiple concurrently executing applications. This could potentially increase the resource usage of the expensive FPGA logic and decrease response times so users will not have to wait for the FPGA to be completely available. The multiple use of an FPGA depends on some form of runtime reconfiguration.

Surprisingly in view of the number of reconfigurable computer (RC) platforms and architectures proposed and built, very few of these projects have included a detailed investigation into run time support. Everybody who ever built a platform has seen the need for a single user loader, often in the guise of interface software between the RC platform and the host system. Some researchers have seen the need for a run time environment such as Brebner [3], Shiraz [4], Caspi [5] and Walder [6]. However as far as we are aware, no-one has actually built an operating system for reconfigurable computing; if the definition of operating system is to extend to allocation of area resources and not just to be a loader of applications. In paper, we present the implementation details of the previously described concept of ReConfigME, an operating system for reconfigurable computing.

The ReConfigME implementation is structured into three tiers consisting of user, platform and operating system which are connected via a standard TCP/IP network (shown in Figure 1). Users connect to ReConfigME through a custom built client interface which enables them to load applications, transfer application data and configuration information, and monitor the reconfigurable computing platform status. ReConfigME enforces a strict FPGA application architecture consisting of a data flow graph structure, memory based I/O, EDIF application file format, and the associated software only components. It supports multiple applications through the use of FPGA hardware resource allocation, application logic partitioning, runtime bitstream generation, and runtime reconfiguration. For easier implementation and due to technology limitations, ReConfigME has a limit on the number of concurrent applications and uses static application memory allocation. The current FPGAs and their design tools do not support dynamic runtime reconfiguration of arbitrary sized applications, thus ReConfigME simulates dynamic runtime reconfiguration. When ReConfigME wants to allocate a new application to the FPGA, all running applications are check pointed, the FPGA clock is

[1] All the authors contributed equally to the work in this paper

1-4244-0054-6/06/$20.00

stopped, and a new bitstream including the new and all the existing applications is downloaded. The existing and new applications are then started or restarted.

This paper is structured as follows. The reconfigurable computing platform used and the factors affecting its selection is described first. In the next section, the restrictions placed on the application's design are outlined as an application architecture. The primitive architecture used to support the inter-process communication abstraction is then detailed. In section 5, ReConfigME's software implementation structure is described which includes the use of a three-tier networked communication architecture. A detailed listing of the procedure involved in executing an application under ReConfigME is then described through the use of a sample application. The applications that were implemented to test the correct functionality of ReConfigME are then detailed. The paper is then finalised with a summary of outcomes in the conclusion.

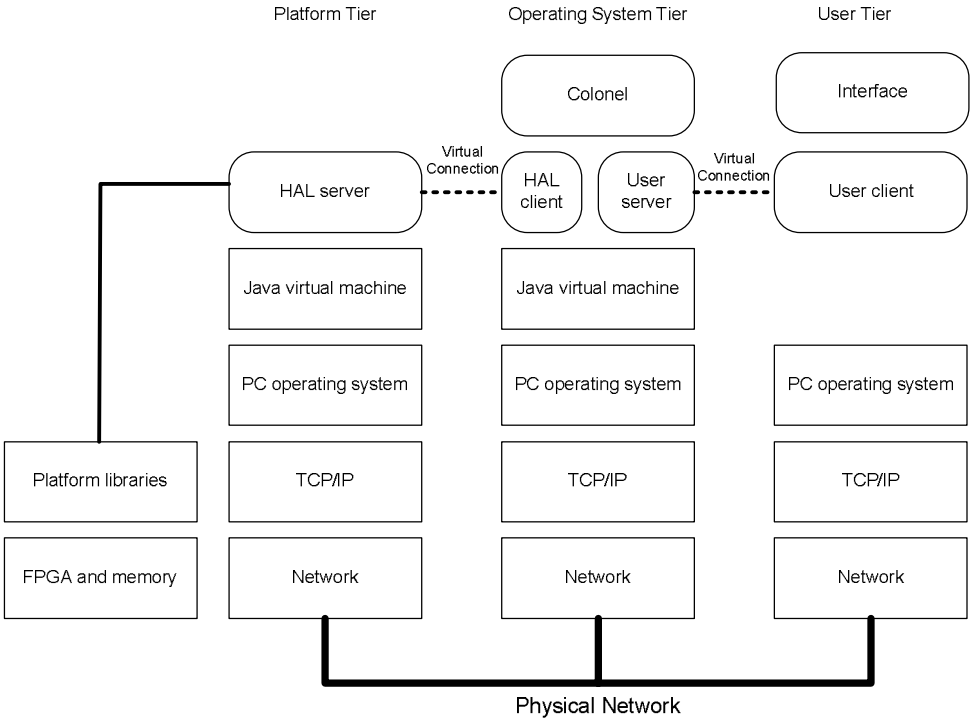

Figure 1: ReConfigME implementation architecture

2. Hardware Platform

The prototype of ReConfigME was developed on a standard PC with a Celoxica RC1000pp development board, in a typical co-processor configuration. The RC1000pp is a standard PCI bus card equipped with a Xilinx Virtex XCV1000 part with 1 million system gates. It has 8Mb of SRAM directly connected to the FPGA in four 32-bit wide memory banks. The memory is dual ported to the host CPU across the PCI bus accessible by DMA transfer or as a virtual address.

This platform was selected for the operating system prototype for several reasons. Firstly, the platform consists of a medium grained FPGA, loosely coupled to a modern high performance microprocessor via a standard PCI bus. The medium grained FPGA has ample resources to be shared amongst multiple concurrent applications and the PCI bus has sufficient I/O bandwidth to support the streaming of data into the applications. Secondly, the platform has four banks of high capacity dual port memory. As was described in the inter-process communication abstraction, processes will communicate with each other and the external microprocessor via the platform's on-board memory. External I/O data will be loaded into a memory bank via the PCI bus and then passed into the process via the FPGA pins and memory controller. This type of I/O transfer requires dual-port memory as the host and FPGA can communicate directly with the memory bank. Finally, the platform supports runtime reconfiguration via SelectMAP over PCI.

3. Application Architecture

The applications used in conjunction with ReConfigME need to be designed with the following four characteristics. Firstly, the applications should be structured according to the data flow graph model shown in Figure 2. This enables the Partitioner to divide the application into several partitions that better match the geometric dimensions of the vacant FPGA

area. However, applications that are not structured as a data flow graph model can still be used with ReConfigME but no attempt to partition them will be made. This may result in an extended application response time, as not being able to partition the application will increase the chance of it being blocked if the particular size of vacant area needed is not available.

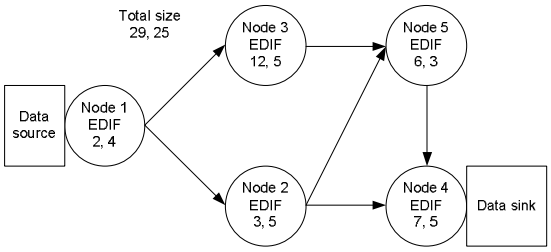

Figure 2: Application Architecture

Secondly, data source and sink nodes are inserted into the applications at points where input or output data is required. As all inter-process communication is conducted via on-board memory, these nodes provide the interface between the application and the on-chip memory controller. The applications in the prototype have access to 1Mb of memory each starting at a virtual address of 0x00. The memory controller will then convert this virtual address to a real address based on the static allocation of external memory. The on-chip memory controller in conjunction with the ReConfigME server is then responsible for reading and writing the data to and from the applications and the appropriate memory location. This interface allows applications to be programmed in both VHDL and Handel-C.

Thirdly, each of the nodes of the data flow graph must be relocatable on the FPGA as ReConfigME will determine where to allocate the nodes at runtime. Since the current tools do not support runtime routing of pre-placed and pre-routed applications, this prevents them from being arbitrary relocated. So each of the nodes of an application is synthesised into an intermediate file format at compile time, but not placed and routed to a bitstream. The intermediate file format chosen in ReConfigME is EDIF. This file format has advantages over many of the others because almost all design entry methods can generate it, it's not commercially specific to an FPGA or company, it has an open source specification, and multiple EDIFs can easily be merged together to result in a single FPGA bitstream. EDIF's are combined by the operating system with an area constraint file that specifies the location of each node and the complete FPGA is then place and routed.

Finally, each of the nodes in the data flow graph model and the entire model itself must have an estimate of the geometric dimensions of the FPGA area they will require when passed into ReConfigME (see Figure 2). Thus it is necessarily at design time to execute the place and route tools over each of the nodes to gain a size estimate. It can be expected that this will not be an entirely accurate area estimate especially if the aspect ratio has to be change and as such a margin for error is added to the area estimate used in ReConfigME.

4. Primitive Architecture

The primitive architecture of ReConfigME is that part of the hardware that is configured onto the FPGA before any user applications and remains there. The primitive architecture is used to support the previously defined inter-process communication abstraction. It consists of a memory controller and network terminators. The memory controller is responsible for granting access to the memory when requested by an application, and managing the transfer of the I/O to the particular application. As the RC1000 consists of four 2Mb memory banks, accessible either via the host computer or FPGA, the memory controller has to negotiate with the platform memory arbitrator to ensure both the host and FPGA applications do not write to the same memory bank simultaneously. For ease of implementation, the memory controller logically divides the memory into fixed sized blocks each of which are then allocated to a single process requiring I/O. Although this limits the total number of processes resident on the FPGA, it will not impact on the results gained from the set of experiments that will be conducted on the prototype.

As I/O arrives at the memory controller from a process, it negotiates with the memory arbitrator to ensure it has exclusive access to the particular memory bank. Once access has been granted, it then has to convert the local addressing scheme that each process is using into the global addressing scheme to ensure the data is loaded into the correct location in memory. The memory controller then either reads or writes the data into the calculated memory position.

Each of the processes is connected to the memory controller via a single network terminator. The network terminators simply provides the matching interface for the data source and sink nodes so processes can easily connect to it. This currently consists of a custom bus of 21 address lines, 32 data lines, 4 single bit control lines, and a single bit clock line. Processes can then either read or write to anywhere within the range up to 1 Mb which is allocated to it.

5. ReConfigME Implementation

The overall architecture of the operating system is component based with each operation separated into

small independent components which communicate via a simple message based mechanism. As there are many issues relating to reconfigurable computing operating systems that have not been fully researched, this type of architecture was chosen over the more traditional monolithic operating system architecture. As the requirements of a modern traditional operating system have been well defined, the implementation of a monolithic architecture is relatively straightforward. However in a reconfigurable computing operating system the requirements are still unclear and as such the construction of a monolithic architecture had to be avoided because they are difficult to maintain as the detailed requirements emerge.

A simple multi-client server arrangement to structure the inter-component communication was chosen to be used with the prototype. This involved one client server connection between the user and bitstream generation components, and another between the bitstream generation components and the reconfigurable computing platform. This allows the user to be remotely located from the majority of the operating system components, possibly via a remote web front end, and the reconfigurable computer to be remotely located from the bitstream generation tools. Another benefit of this design is that ReConfigME can manage multiple FPGA cards which can be physically located within the same machine or in separate machines making it easily scalable. Such an arrangement allows maximum flexibility with respect to location of the user, platform and ReConfigME's bitstream generation components.

The inter-component communication structure has ReConfigME divided into three tiers; user, operating system, and platform. Although there is no general agreement about what contributes as a tier [7], a machine separated by network communication is considered a tier in this prototype. The client tier primarily performs the interaction between the operating system tier and the user by providing a shell as an interface. The operating system tier contains the operating system architecture that consists of the resource allocation, application partitioning and bitstream generation. The platform tier consists of the reconfigurable computing platform and the components needed to access it. ReConfigME's components were then separated into these three tiers and can be seen in Figure 1. The curved cornered rectangles indicate the component was constructed specifically for the prototype. Rectangular components represent off the shelf products. This figure is very similar to a protocol stack; data enters the tier via the bottom component which is connected to the others via a physical network. Data progresses through the tier until it reaches the destination component. Likewise data that needs to be transferred to another tier will progress down through the tier until it reaches the physical network. Each of the components and tiers will now be discussed in more detail.

5.1. Platform Tier

The platform tier consists of seven components and is primarily responsible for the communications to and from the reconfigurable computing platform. All of the components except the reconfigurable computer and network are all resident in software on a PC that hosts the reconfigurable computing platform. The top level component is the hardware abstraction layer (HAL) server and is responsible for hiding the platform specific API. It is a simple API written in Java that can be used with various platforms to offer access and control over the hardware. It provides methods for reading and writing bitstreams to the FPGA, reading and writing to the on-board memory, and clock management. As the RC1000 used in ReConfigME is shipped with C++ libraries, Java native method calls were used to connect the hardware abstraction layer API to the corresponding RC1000 library method. The advantage of the hardware abstraction layer is the same API can be used to communicate to any number of different target platforms.

The hardware abstraction layer also supports a client/server paradigm so the reconfigurable computing platform can be remotely located. Connections are made to the HAL server via standard TCP/IP sockets from the HAL client, located in the operating system tier. Bitstream files, input and output data, and clock configurations are then passed back and forth between the client and server.

The other components in the platform tier are used to support the HAL server. Java was chosen as the implementation language because of its ease of internetworking, its object orientated semantics, and its portability across different hosts, operating systems, and hardware. The PC operating system component, in this case Windows XP, is needed to manage the hardware resources of the host computer and the TCP/IP and network components are required to provide the connectivity between the HAL server and the HAL client.

5.2. Operating System Tier

The operating system tier consists of seven components and is responsible for allocating and partitioning applications, the generation of the FPGA bitstreams, and the transfer of application data and configuration information between the platform and user tiers. The top level component of the operating system tier is dubbed "*Colonel*" (analogous to a software operating system but is spelt differently to avoid confusion). The Colonel does everything except

the transfer of data between the other tiers. It consists of three sub-components and the bitstream generation tools.

As a user connects to ReConfigME, their application and configuration information is passed into the Colonel via the user server. The application and its pre-compiled geometric dimensions are then passed onto the Allocator and in conjunction with the Partitioner, will determine whether the application can configured onto the FPGA or is blocked and put into a queue because of the lack of vacant area. The Allocator consists of the Minkowski Sum with bottom left fit algorithm that was described in section [8] and the Partitioner consists of the modified temporal partitioning algorithm that was described in [9].

Once all the locations of the application's partitions have been determined, the Allocator will create a file which ensures the application's absolute placement details calculated by the Allocator are followed once the FPGA bitstream is generated. In ReConfigME, the constraints file is in the standard vendor format. The main control loop will then create and call a script that executes the place and route tools. This will generate an FPGA bitstream that includes all of the loaded applications in their correct locations. It is then passed onto the HAL client who is responsible for connecting to the platform and configuring the new bitstream onto the FPGA.

The Colonel also manages the transfer of application data involving capturing the input data from the user loading it into the on-board memory, and reading the output data from the on-board memory and passing it back to the user. This task primarily consists of an address translation. The local addressing scheme is translated into the platform's global addressing scheme to ensure the correct location in the platform's memory is accessed for either reading or writing. The Colonel also passes specific clock and platform information between the HAL client and user server.

The second level component of the operating system tier is the HAL client and user client. The HAL client component is responsible for creating a connection to the desired platform and passing all of the I/O, bitstreams and configuration information between the two. It allows the platform to be remotely located from the Colonel. The advantage in this is ReConfigME can target numerous different platforms without having to have them all located in the same machine as the Colonel.

The user server handles all the communications between the user client in the user tier and the Colonel. This includes input and output of application data, incoming applications, and platform configuration information such as clock settings. The user server accepts connections via standard TCP/IP sockets from numerous remotely located clients located in the user tier. Once a connection has been established, it is responsible for passing the data to the Colonel and then responding to the client with the associated response. The advantage of having the communication component separate from the Colonel is if the network protocol or client/server API is altered, only those components need to be modified, not the complex Colonel itself.

5.3. User tier

The user tier contains five components and is primarily responsible for providing a user interface and connection to the operating system tier. The top level component is the user interface and consists of a combination of a simple command line interface for user input and a graphical user interface for displaying the geometrically layout of currently executing application on the reconfigurable computing platform. Via the command line interface, users are able to load applications, stream I/O data to the platform's on-board memory and configure particular platform settings such as clock values. The graphical user interface displays the results of the allocation and partitioning of applications as they are loaded into ReConfigME.

The user client is the second level component in the user tier and provides an interface to the Colonel via the user server. It communicates via standard TCP/IP sockets to the user server located in the operating system tier and simply converts user requests from the command line interface into the API defined for use between the user client and server components. The advantage in using the user client and server is other user interfaces can easily be added with little or no change to the Colonel.

6. Sample application execution

There are two types of files that are needed to be created for an application to be loaded onto the reconfigurable computer via ReConfigME: the application itself with an EDIF file for each data flow graph node, and a Java class file that defines how each of these EDIF files are connected together in data flow graph model. The first stage in developing an application for use with ReConfigME is the generation of the series of EDIF files that describes the behaviour of the application. This procedure initially involves the designer determining how the application will be structured.

Each of these nodes will then result in a single EDIF file. Almost any design entry method can be used to create these nodes but in this example the hardware description language developed by Celoxica known as Handel-C was used. Shown in

Figure 3 is a code listing of the first node in a sample application.

It simply reads a 32 bit number from the first location in memory, adds one to the number, and then writes the result back into the second location in memory. As can be seen from the code, the data is loaded into the memory from the host via the readMem() and writeMem() methods.

```
set part = "V1000BG560-4";
set family = XilinxVirtex;

#include "taskInterface.hch"

void main(void) {
    while (1) {
        // Read from address 0 until the value is 1
        unsigned 32 data;
        data = 0;
        data = readMem (0);
        while (data != 1) {
            data = readMem (0);
        }

        // Read from address 1
        data = readMem (1);

        // Add 1 to the result
        data += 1;

        // write the value back to address 1
        writeMem (2, data);

        // Write 0 back to the control register telling the host
        // that processing is done.
        writeMem (0, 0);
    }
}
```

Figure 3: Handel-C code listing for add one data graph flow node

These methods insert the data source and sink nodes into the application so it can be connected to the memory controller. The Handel-C source code for the other nodes in the data flow graph look very similar except instead of adding one to the number, the second node performs a logical XOR against a set mask and the third node performs a logical AND against another set mask. All the Handel-C source files are then compiled and an EDIF file is generated for each node. As is shown in the code in Figure 4, three new cores and their dimensions which represent each node in the graph are added into the instance *tg*.

```
// set up data flow graph
TaskGraph tg = new TaskGraph();
tg.setPrefDimensions(3,6);
tg.setName("applicationTest");
tg.setIsPartitionable(true);


// add the nodes onto the graph
tg.addVertex(new Core(1 ,1, "add_one_core"));
tg.addVertex(new Core(1, 2, "XOR_core"));
tg.addVertex(new Core(2, 2, "AND_core"));


// add edges or communication links on the graph
tg.addEdge(0, 1);
tg.addEdge(1, 2);
```

Figure 4: Java class file defining data flow graph structure

The code initially involves creating an instance of the class *TaskGraph* which represents a data flow graph, and initialising the parameters defining its geometric dimensions, name and whether it should be partitioned. An application can be prevented from being partitioned by ReConfigME if the designer believes it has strict performance constraints. Each of the EDIF filenames and the area they will consume are then added into the structure of the data flow graph as nodes by simply adding a new Vertex into an array within the instance. In this sample execution, the data flow graph consist of three nodes or EDIF files; add_one_core, XOR_core, and AND_core. The edges which represent the communication links between the nodes in the data flow graph are created in the instance by calling the method *addEdge* and passing the core numbers of the communicating nodes. In the sample application, the add_one_core connects to the XOR_Core, which connects to the AND_core.

The next part in the Java class file is to define the connection to the ReConfigME server and pass the TaskGraph object containing all the data flow graph details. This is simply performed by creating an instance of the class RC1000 with the parameters of the TaskGraph, IP address and port number of the server. This results in the instance of the data flow graph and all associated EDIF files being loaded into ReConfigME so the generation of the bitstream can begin. Once the bitstream has been generated and dynamically configured onto the FPGA, the necessary read and writes to and from the memory are performed.

The final stage in the sample application execution is to read and write the I/O data with the output data being stored in a local file. The status of these actions is reported via the client interface. Once the client disconnects from the ReConfigME server, the application is removed from the FPGA and the new bitstream is generated and configured onto the FPGA.

7. Applications for ReConfigME

There were two main applications implemented to verify the correct functionality of ReConfigME. Two applications were implemented to show that ReConfigME can be used with real applications that require large amounts of I/O to be transferred between the hardware circuit and software part of the application. These two applications are described here.

7.1. Blob tracking

Blob tracking is a term used in the vision tracking research community which is the process of finding the location of a known object in a series of images. In the application described here, the object of interest is an orange coloured ball and a series of images were taken as the ball was randomly moved. The first step in blob tracking algorithm implemented for ReConfigME was to separate the orange coloured ball from the rest of image. This is achieved by performing a threshold operating on the image, based on a colour value that

matched the orange ball. Each pixel in the image was examined to determine if it matched the colour of interest. If the pixel matched the colour, in the output image it was set to white whereas if it did not match, it was set to black. This procedure was repeated for every pixel in a frame. Once the known colour had been separated from the image, the centre of these pixels had to be calculated. This was achieved by simply calculating the mean location of all the pixels that matched the threshold colour of interest. This point was then indicated by the use of red crosshairs.

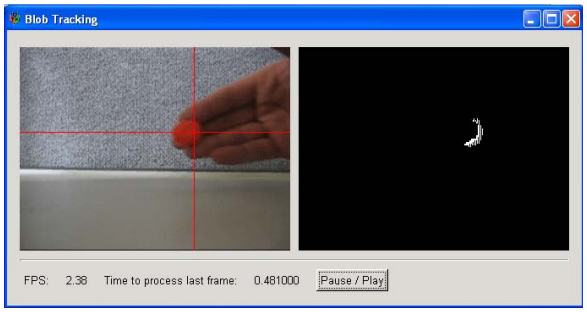

Figure 5: Blob tracking executing on OS

The application consists of two parts: the hardware circuit containing the blob tracking algorithm which performs the threshold and calculation of the centre location written in Handel-C, and the software application responsible for transferring the I/O to ReConfigME, capturing the video in real time via a camera, and displaying the threshold image and location of the crosshairs. The hardware circuit of the blob tracking application consumes approximately 400 CLBs or 7% of the target FPGA, has pre-defined dimensions calculated to be 20 CLBs by 20 CLBs and is 70 lines of non-commented Handel-C.

Edge enhancement is another well-known image processing algorithm and involves identifying the edges of objects in an image. This algorithm is often the first stage in template matching or target recognition. The algorithm firstly involves performing a threshold of the intensity change across a window of pixels. If the intensity change exceeds the selected threshold, the pixel is marked as an edge. The window is moved across the entire image in both a horizontal and a vertical direction. The output from the edge enhancement application is shown in Figure 6.

Figure 6: edge enhance executing on OS

As was the case in the blob tracking application, the edge enhancement application consists of two parts: the hardware circuit that executes the edge detection algorithm, and the software part that transfers the I/O to ReConfigME and displays the resultant edge detection. The hardware circuit consumes 480 CLBs, has pre-calculated dimensions of 40 CLBs wide by 12 CLBs high and is 111 lines of non-commented Handel-C code.

7.2. Multiple concurrent applications with ReConfigME

Shown in Figure 7 is the allocation status when both the blob tracking and edge enhancement applications were allocated onto the FPGA at the same time from the Xilinx Floor-planner. The blob tracking application is shown in yellow, the edge enhancement in green and the memory controller in light grey. This figure reflects the allocation constraints placed onto the applications by ReConfigME.

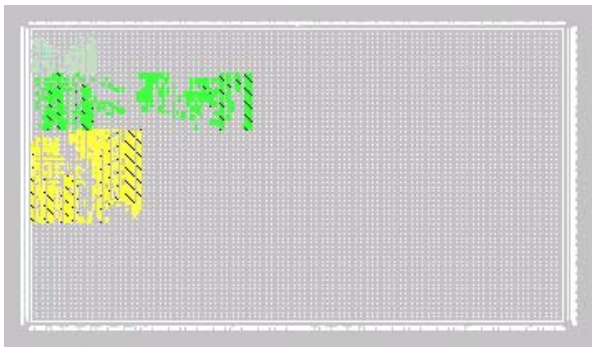

Figure 7: Xilinx Floorplanner

With both the applications allocated onto the FPGA and the clock set to 25MHz, both applications executed correctly and there was no noticeable difference in the frame rate of both applications as compared to running them separately. The output from both applications was identical when compared to the output generated when each application had exclusive use of the FPGA.

The edge enhancement application was removed by the operating system, the network terminator was re-allocated, and the blob tracking application continued to execute correctly. Finally, the blob tracking application was removed from ReConfigME and the FPGA was re-configured with no applications.

8. Conclusion

In this paper a prototype operating system known as ReConfigME which is based on the architecture previously described in [10] was presented. This included details on the selected platform and the detailed implementation. It was discussed how the applications are created so they are structured according to the data flow graph model, the primitive architecture that is used to support inter-process communication, the networked tier architecture used to implement the prototype itself, and sample application execution listing. Two applications that were written to demonstrate the OS were also presented.

9. Acknowledgements

The authors would like to acknowledge the financial support from the Sir Ross and Keith Smith Trust Fund. They would also like to acknowledge contributions from both Martyn George and Maria Dahlquist.

10. References

[1] Celoxica, "RC2000 Hardware Reference Manual," 2003.

[2] Bioler 3 Reconfigurable Computing Platform Datasheet 04.

[3] G. Brebner, "A Virtual Hardware Operating System for the Xilinx XC6200," presented at 6th International Workshop on Field-Programmable Logic and Applications (FPL'96), Darmstadt, Germany, 1996.

[4] N. Shirazi, W. Luk, and P. Cheung, "Run-time Management of Dynamically Reconfigurable Designs," presented at International Workshop on Field-Programmable Logic and Applications (FPL'98), Tallinn, Estonia, 1998.

[5] E. Caspi, M. Chu, R. Huang, J. Yeh, Y. Markovskiy, A. DeHon, and J. Wawrzynek, "Stream Computations Organized for Reconfigurable Execution (SCORE): Introduction and Tutorial," presented at 10th International Workshop on Field-Programmable Logic and Applications (FPL'00), Austria, 2000.

[6] H. Walder and M. Platzner, "Non-preemptive Multitasking on FPGAs: Task Placement and Footprint Transform," presented at Engineering of Reconfigurable Systems and Algorithms, NV, USA, 2002.

[7] U. Amjad, *Application (Re)Engineering*. Upper Saddle River, New Jersey: Prentice Hall, 1997.

[8] G. Wigley and D. Kearney, "The Management of Applications for Reconfigurable Computing using an Operating System," presented at 7th Asia-Pacific Computer Systems Architecture Conference, Australia, 2002.

[9] G. Wigley and D. Kearney, "Research Issues in Operating Systems for Reconfigurable Computing," presented at Engineering of Reconfigurable Systems and Algorithms (ERSA02), Las Vegas, USA, 2002.

[10] G. Wigley, D. Kearney, and D. Warren, "Introducing ReConfigME: An Operating System for Reconfigurable Computing," presented at Field-Programmable Logic and Applications, Reconfigurable Computing Is Going Mainstream, 12th International Conference, FPL 2002, Montpellier, France, 2002.