UDP Partial Bitstreams Diffusion Through WLAN

Jeremie Crenne, Pierre Bomel, Guy Gogniat, Jean-Philippe Diguet
LabSTICC Laboratory CNRS UMR 3192 Universite de Bretagne Sud UEB Lorient, France
{jeremie.crenne, pierre.bomel, guy.gogniat, jean-philippe.diguet}@univ-ubs.fr

***Abstract*—In this paper we present the use of UDP through WLAN to perform partial bitstreams downloading based on a Xilinx Virtex V2p FPGA. Introducing a simple, standard, with a low software and hardware memory footprint, as well as a good transfer rate protocol, is a great way to promote dynamic reconfiguration on a large field of embedded equipments where real time constraint could be a bottleneck. Our approach targets low cost embedded handled systems, robotics devices, or even at a more large scale, spacecraft applications because of the reduction in terms of price, weight and power consumption. Real implementations measures bring our design up to a sustained 30 Mb/s rate on a WIFI link. Minimal architecture leads both software and hardware to a quite low memory cost, respectively 200 KB and 2524 slices, be a total utilization of 8% of the programmable chip.**

***Index Terms*—Dynamic partial reconfiguration, FPGA, LAN, Ethernet, WIFI, UDP,**

I. Introduction

In telecommunication industry, reconfigurable wireless Universal Terminal is now a well known idea which first appears in military area and became civilly popular in the 90s. This growing topic is a direct consequence of increasing performances of FPGAs (Field-Programmable Gate Array). This technology enables massive parallelism, enough computational power to realize DFE (Digital Front End) and the capability to be reconfigured at moderate power consumption. Assuming that a device should support several digital mobile telephony services, digital broadcasting services, and/or digital data transfer services, it can take advantage of the partial reconfigurability. Current devices impose severe limited services due to inflexibility of their analog technology parts but tend to be bypassed by Software Defined Radio. SDR is a set of techniques that allows reconfiguration of a communication system without the need to physically change any hardware element. Underlying goal is to produce devices capable of supporting different services (multi-standard) with an adaptation of their hardware components in function of the wireless network such as GSM, GPRS, UMTS and WIMAX. In addition, they should be able to deal with wireless LAN standards like IEEE 802.11 known as WIFI. Delahaye and al. [DGRB04] prove the feasibility of dynamic partial reconfiguration on a heterogeneous SDR platform which provides a flexible way to build highly reusable systems on demand. Such devices are requiring to dynamically adapting a subset of their functions in order to take all the variations in "real-time". Thus these systems can take advantage of the dynamic partial reconfiguration (DPR) by swapping hardware resources on demand.

When talking about partial reconfiguration, additional resources are required to store an increasing number of partial bitstreams. Flash and external RAM memories are mainly proposed by researchers as repositories. Where the first one is clearly dedicated when non-volatile configurations are considered, the second can be only interesting for volatile ones. We are facing the migration of silicon from FPGAs to memories. Their low cost compared to FPGA is in favor of such change but raises some drawbacks:

1) Memory reuse rate can be low, and could be used only one time, on boot.
2) For a single function to implement, the space of possible bitstreams is large and could become bigger than local memories. This is due to FPGAs types (numerous devices, families, sizes, packages), possible configurations depending on IP' shapes/placements and the natural commercial life of IPs that regularly introduces new versions and updates.

In this context, we have previously addressed this issue by introducing a three-level server's hierarchy where a LAN remote server is presented for storing FPGA's configuration [BDGC08]. The server could be directly wired to offer best performances over the market. For nomadic devices, it is not very advisable nor conceivable as it is considered as a hard physical constraint. A better approach is to remove this restriction and thanks to popular wireless connection networking WIFI as known as IEEE 802.11 standard, it is possible at low cost.

In the following we describe in Section 2 previous works on standard protocols and partial reconfiguration related implementations into a LAN. Then in Section 3, we detail our choices for optimum bitstream diffusion through a local network. In addition, in Section 4 we describe our contributions in terms of software and hardware architectures. Section 5 is intended to cover our experiments and measures. Finally, we conclude and open the discussion for further improvements and extensions we intend to focus on in a short-term.

II. Related Work

A. Common Use of Transport Protocols

Rind and al. [RSQ06] describe choices for TCP (Transport Communication Protocol) over UDP (User Datagram Protocol) and vice-versa related to speed, numbers of mobile devices and link capacity (bandwidth) metrics. Results are given in terms of throughput and goodput via a network simulator. It shows that TCP is giving better performance when minimum number of mobile devices are connected to a WLAN (Wireless LAN) and clearly setup that faster moving nodes are highly disturbing packets transmissions. UDP is found better if it is possible to bear little loss of packets. Consequently it is a first choice protocol for fast delivery of data. Uchida [Uch07] presents an hardware-based TCP processor for Gigabit Ethernet which requires only one Ethernet PHY device. The circuit is small enough to be implemented on a single FPGA, with an announced 949 Mb/s throughput in both emission and reception directions. It has to be carefully compared with other systems. With this approach, no high traffic over the network is considered and TCP congestion control is well known to be designed and optimized for wired networks. Moreover, as the system we target is using a WIFI link an thus is limited to a much lower throughput, very high Gigabit transfert rate is oversized. Ploplys [PA03] and al. perform a study where ”wireless” UDP is used for real-time performance in control. Loss of data is well defined, explained and evaluated based on many factors such as range, environmental obstacles, computational loads and increased network traffic. Existing work establishes that TCP is vastly employed in WLAN topology and UDP in wireless. The use of UDP is natural when targeting wireless handled devices. UDP is also the most suitable standard for systems with a high latency and need by nature, a shorter communication time.

B. Dynamic Partial Reconfiguration Discussion

Compton and al. [CH02] give a complete survey of the whole problematic about reconfigurable computing and agree that PR latency is one of the most critical aspects in its implementation. Reconfiguring time is highly dependent upon the size and organization of partial reconfiguration region inside the FPGA. Partial bitstreams in latest Virtex V4 and V5 are 2d shaped and relaxed concerning the column constraint introduced before with previous technologies. It results in a decrease of occupied reconfigurable area into the FPGA. This area size is measured in frames of 41x32-bit words and is increasing on larger devices.

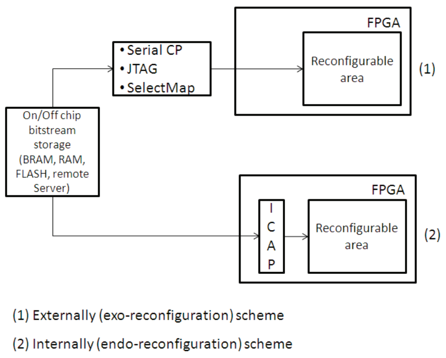
•Serial CP
•JTAG
•SelectMap
FPGA
Reconfigurable area
(1)
On/Off chip bitstream storage (BRAM, RAM, FLASH, remote Server)
FPGA
I C A P
Reconfigurable area
(2)
(1) Externally (exo-reconfiguration) scheme
(2) Internally (endo-reconfiguration) scheme

Fig. 1. Partial reconfiguration methods

From Xilinx’s datasheet, two types of partial reconfiguration exist (Figure 1). Partial and dynamic reconfiguration of Xilinx’s FPGA can be done with an Internal Configuration Access Port (ICAP) [Xil06a]. Virtex V2p, Virtex V4 and V5 series contain this port and can be interfaced with hardware IP or hard/soft processor core such as PowerPC, Microblaze or OpenRISC. Maximum downloading speed rate announced by the fabless in internal reconfiguration mode is capped to a maximum of 800 Mb/s when ICAP accesses are 8 bits wide. Entire costs for hardware implementation is 150 slices and only a single BRAM. Claus and al. [CZMS07] propose a real-time video application demonstrating dynamic partial reconfiguration because of the adaptative nature of the image processing flow. The platform is a V2 where a PPC405 executes the software without a running operating system for managing the reconfiguration. Unfortunately, no speeds measurements were provided at publication time. Lagger and al. [LUSG06] propose a system (ROPES) dedicated to the acceleration of cryptographic functions. The FPGA is a V2 1000 with a synthesizable Microblaze processor. uClinux is used as RTOS and the software is downloading bitstream using HTTP and FTP protocols. PR latencies are respectively ranged from 240 Kb/s for HTTP and 480 Kb/s for FTP. William and al. [WB04] have developed a Clinux device driver on top of the ICAP, enabling bitstreams downloading. The system is based on a Virtex V2p and articulated around a synthesizable Microblaze processor which is running the executable. As no measures are provided, estimations done in a similar context, lead to a transfer speed ranging from 1.6 Mb/s to 3.2 Mb/s. This shows that ”TCP + uClinux” is widely accepted as a universal platform. Anyway we underline the low bitstream downloading performance issue with such a couple, which is difficult to be envisaged in a real-time DPR application. Best downloading speeds described are far below what the ICAP and network bandwidth. Certainly usefull for fast development, uClinux can not be considered with a highly time-constained environnement. In a previous work [BCY$^+$09], we propose an implementation with a ad-hoc data-link level protocol over a standard Fast Ethernet LAN dedicated to lightweight dynamic partial reconfiguration. It is implemented on a Virtex V2p which is running a PPC at 100 MHz, without any RTOS. A sustained 80 Mb/s is reachable with a very little amount of software memory (Table I). In this paper we extend this work where we had introduced a specific and fast downloading of partial bitstreams over Ethernet in a wired LAN. Where our protocol has been found useful when no IP routing is required and only a little amount of hardware and software resources is available, the use of UDP fits perfectly with ”over-the-air” wireless devices, subject of this paper.

	[LUSG06]	[WB04]	[Xil06b]	[BCY$^+$09]
Throughput (Mb/s)	1.7	3.2	4	80
Memory (bytes)	$> 1M$	$> 1M$	$< 100K$	$< 100K$

TABLE I
COMPARATIVES THROUGHPUTS AND SOFTWARE MEMORY FOOTPRINTS

III. Bitstream Diffusion through LAN

We discuss in this section about several bitstream diffusion protocols and what are pros and cons of each. We first quickly come back to the well known TCP/IP architecture model and then take some time to analyze three heterogenous protocols: data link ad-hoc protocol, TCP and UDP.

A. TCP/IP Architecture Model

TCP/IP [RFC89] (Transmission Control Protocol/Internet Protocol) is a networking reference framework used for developing networking applications. This model is usually described as a four-layered architecture as shown in Figure 2. For a communication, data are sent from layer 4 to layer 1, and vice-versa for a reception. In the following lines, we place the discussion on layer 1 and 3.

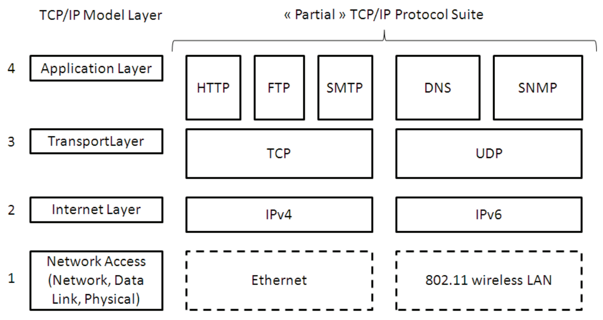

Fig. 2. The five-layered TCP/IP architecture model

B. Network Access

The TCP/IP Network Access layer (layer 1) is concerned with physical characteristics of the transmission medium. Ethernet standardized IEEE 802.3 [Eth] is a family of computer networking technologies for local area networks with a number of standards: 10 Mb/s Ethernet, Fast Ethernet 100 Mb/s, Gigabit Ethernet 1 Gb/s and 10-Gigabit Ethernet 10 Gb/s. To enable communication with multiple clients, it uses a coaxial cable as a shared medium in order to attach every system and the way the devices are sharing the channel is governed by the CSMA/CD algorithm. To obtain a greater transmission length and simplifying cabling, Ethernet repeaters are encouraged. With todays hubs, transmission error rates are very small and shows they are in favor of very simple error detection and recovery strategy. A restart at bitstream level is then possible rather than at packet level. International IEEE 802.11 [WIF] describes characteristics liable to a wireless LAN (WLAN). It makes possible to build broadband wireless local networks and in practice allows to link computers, laptops, PDAs, communicating devices and other peripherals, indoor or outdoor with ranges, speeds and modes depending on numerous revisions of the standard from 802.11a to 802.11s. Wireless is sometimes the only possibility in applications where it is required to have a great mobility. In industry, reduction of wiring proves its pertinence: costly to install, to repair, imposing strict placements limitation. Compared to Bluetooth, the WIFI main strength stands in its higher throughput and range. When chosen, an ad-hoc protocol is intended to produce a low level implementation of the minimal layer, abstracting accesses to the involved hardware resources and the use of a specific data-link (embedded into layer 1) level protocol. This generally gives higher transfer rates as communication doesn't go through all netwoking layers. Anyway, the main drawback is the downloading of bitstreams which is then not possible from any machine over a network. That is why and mainly for industrial and homogenous reasons, it is not viable. We propose to migrate to a more commodious protocol, that can be found in transport layer.

C. Transport Layer

Today, IP protocols are adapted to low latency and high bandwidth data transfers. Therefore, this second point leds us to adapt our protocol to one of the most common transport protocol (layer 3). TCP [RFC81] is used in many non-critical applications such as HTTP, FTP and SMTP. It is connection-based and guarantees that the receiver will gets exactly what the sender sent without any errors and in correct order. TCP resends every packet if it doesn't arrive correctly to the destination. To avoid congestion, TCP is cutting down speed whenever a packet is lost and re-increasing it slowly when packets are successfully emitted. As for the most appropriate transmission protocol, we pinpoint that packets looses in TCP are attributed to congestion i.e a higher traffic. Hence for a wireless environment where bit error rate is high, TCP performances are highly degraded due to its window based congestion control mechanism. UDP [RFC80] is similar to TCP and stands in the same TCP/IP layer. Known UDP applications are DNS and SNMP. Connectionless, its difference is located in the relationship between two parties. In other words, one can send data to another and that is all. UDP doesn't provide any reception reliability thus, there is no guarantee that packets will arrive. However if the transmission is correct, the packet will be received without any data corruption. UDP is faster than TCP as there is no extra overhead for error-checking above the packet level. A comparison between TCP and UDP is given in Table II. With this table and previous studied arguments, UDP has been found to be the most acceptable protocol for WLAN bitstream diffusion. From this point, we assume the use of UDP over the network.

Protocol	Complexity	Speed	Architecture	Caveats
UDP	Low	High	Broadcast or Client/Server	Unreliable, String data
TCP	Average	Low	Client/Server	String data

TABLE II
Comparison of TCP and UDP protocols [Ins09]

IV. Contribution

We present in this section our contribution in details. Software and hardware architectures are both developed to show precisely how works our protocol. This contribution does not depend on the targeted FPGA as tests (on V2 and V4 implementation) show us that speeds are close. The discussion is placed on the architecture rather than on the latest FPGA version.

A. Software Architecture

Our architecture workflow on Figure 3 shows that we have to differentiate the software running onto the server and the software onto the FPGA. On the left hand, the server executes a console application eventually hooked to a front-end executable. On the right hand, onto the client FPGA, the processor runs an executable build with a PPC GNU GCC and a TCP/IP stack.

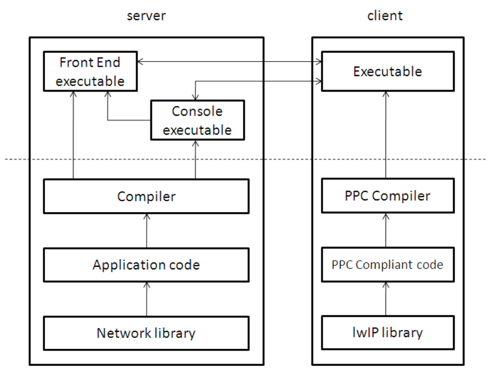

Fig. 3. Server/client software workflow

1) lwIP as a TCP/IP networking stack: Instead of developping a networking library from scratch and in order not to reinventing the wheel, we have chosen an open source TCP/IP stack designed for embedded systems: lwIP. Directly available in EDK, lwIP [Dun01] is an implementation under BSD licence of the TCP/IP stack with RAM usage friendly in mind. It was initially written by Adam Dunkels of the Swedish Institute of Computer Science and is now maintained by several developers headed by Leon Woestendberg and hosted by Savannah. The porting proposed by Xilinx in EDK is quite robust (both Microblaze and PPC can be used without any problems), with a lot of parameters configurable at compile time that can be tuned to tailor an architecture according to the requirements. lwIP is also featuring a quite exhaustive characteristics list (IP, ICMP, ARP, UDP and TCP) and can be run with an underlying OS or not.

Our first approach was to tailor lwIP to use UDP only as we don't need another protocol. To ensure packets producer-consumer paradigm, lwIP stack uses a pool of buffers. This pool is a critical point in terms of performances and goodput and has to be well scaled. Default setting value for this segment is close to 800 KB large be 512 packets of 1528 bytes. With that consideration we have found native lwIP parameters to be oversized in EDK. The absence of transmission errors during days of testing, which consist of sending and receiving data as fast as possible and checking right packet order, proves that reducing it to 100 KB is pertinent without interfering overall performances. Next, we have tried to unload the processor which is running the software from heavy computations. To achieve this goal, an option called UDP checksum offload could be set. It enables the network adapter to compute the UDP checksum on transmit and receive, saving the CPU from having to do it. The gain in term of throughput is significant (x2) when the PPC is clocked at 100 MHz and can't handle too much computation.

2) Abstracting lwIP: We have developed a 800-line of code software framework for abstracting lwIP functions calls as much as possible. This is not only to ease the use of the stack, but also to make fast porting in the future. With this library, a new developer is able to pay his attention not onto how to make the lwIP works with UDP protocol, but only to use it without any additional knowledges. The full set of functions comes documented, well understandable and is not platform dependant, thus it is portable without changing a line of code.

3) Software DPR Protocol: We are now describing our protocol by first introducing the chosen frame's structure. It is always constituted of two fields as shown in Figure 4. The first one is reserved to special purpose such as session code id or frame number and is 4 bytes long. The second is the data, i.e. the bitstream.

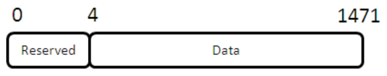

Fig. 4. Frame structure

The allowable packet size relies directly to the frame format we use: DIX Ethernet Version II frame format. The total minimum size of one Ethernet frame is 1538 bytes. It includes 12 bytes of inter-frame spacing and 8 bytes of preamble. From these 1518 bytes, 4 are for Frame Check Sequence (FCS). Another 6 are for destination Ethernet address, 6 are source Ethernet address, and 2 are the type, leaving 1500 bytes for user data in each frame. Lastly, 20 bytes go to an IP header, and 8 to the UDP header, leaving a maximum of 1472 bytes for data in each frame. Bitstreams are generally bigger than this maximum so it has to be fragmented before emission. Packet's transmissions are synchronized using a classic end-to-end handshake for ensuring correct data transmission (Figure 5). P Packets can be actually sent from the server depending on what is called the burst size. When P = 1, only one packet is dispatched before receiving an acknowledgement from the client. If P = 5, five packets are transmitted without any data flow control feedback.

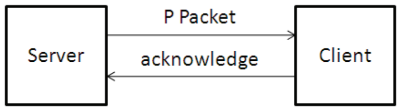

Fig. 5. Protocol communication model

Next we have introduced a process between server and client called session. Session is used to know that one, on the server or client side, is trying to communicate with the other, or is already in communication. A session is dedicated to one user only and has two states: closed or opened. It can be only opened with a code id, and can only be closed when process is finished or an error (or timeout) occurred. Two types of sessions are implemented: bandwidth test and partial bitstream diffusion. When receiving a session code BITSTREAM, the server is looking to initiate a transfer of a given bitstream to the current client. The protocol is able to work in two modes: slave and master (Figure 6). In master mode, the FPGA is responsible for asking the LAN server a partial bitstream. In slave mode, it reacts to the server requests and is forced to update itself.

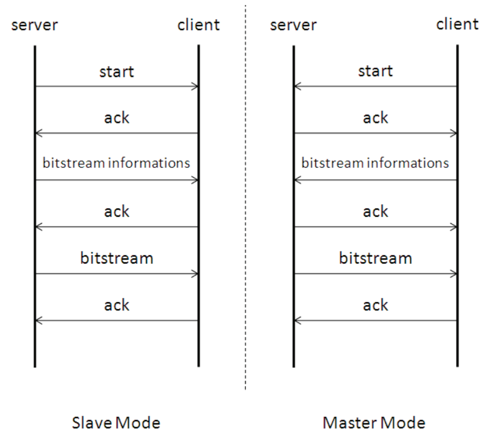

Fig. 6. Slave and master mode for a bitstream session

Throughput is directly depending on the UDP packet size that a device is going to send to another one at a given time. Without tracking tool that measures its behavior, a network is an unknown environment and can change with a large set of parameters such as number of devices or general traffic. Finding out the packet size that our server has to send to get the best throughput is a rather natural way to encompass our goal and simple to implement. Ping is a well known ICMP command to send out an Echo request from one machine to another with a feedback of useful information, such as delay. We mimic it by sending incremental packet size ranging from 4 to 1472 bytes stepped 4 bytes by 4 bytes with an acknowledgment from the client between each transmission. A timer is running on the server during all the session. Between each transaction (Tx/Rx), the timer is scanned to evaluate current throughput. This session can be started with the id code BANDWIDTH TEST.

Obviously, obtaining a maximal reconfiguration throughput must be considered with care. Safety concerning the write of a partial bitstream to the reconfigurable area is necessary in partial reconfiguration context. A loss of a packet will result in an incomplete form of data reception, so on an impossibility of writing the complete partial bitstream into the reconfigurable area. Manifestly, it will lead to an unpredictable behavior. At this time, our protocol only implements a timeout without any retransmission processes. The session is then prematurely closed by the server, and server and client reset themselves and come back to an idle state.

B. Hardware Architecture

As seen in the previous section, our software DPR protocol is simple to implement and to use with our underlying software framework. Concerning, our hardware architecture, Figure 7, it is all done using tools versions that we find mature with a minimum amount of bugs when used with dynamic partial reconfiguration. Both Xilinx's tools EDK, ISE 8.2 as well as PlanAhead 10.1 for the partial workflow are used.

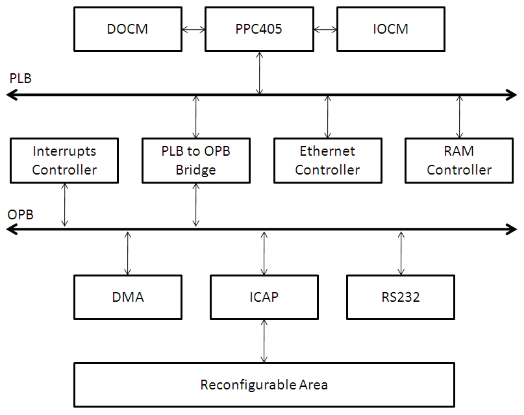

Fig. 7. Hardware architecture

1) Details of Implementation: Our architecture is built around a PowerPC PPC405, hard wired processor running at 100 MHz on a XUP evaluation board from Xilinx. The FPGA part is a Virtex V2p30. Two privates' memories to the PPC DOCM (Data On Chip Memory) and IOCM (Instruction On Chip Memory) are used for holding some program and other data as well as two buses to enable communication between components: PLB (Processor Local Bus) for faster IPs and OPB (On Chip Peripheral Bus) for slower ones. 64 bits PLB bus, provides a high data transfer rate potential with bandwidth capabilities up to 800 Mb/s at 100 MHz. OPB is a 32 bits data length bus. This leads to a 400 Mb/s bandwidth when clocked at 100 MHz. Ethernet controller is connected to the PLB, so is the RAM (Read Only Memory). The INTC (Interrupt controller) is also added to the design and linked to the OPB as well as a DMA (Direct Access Memory) for performance measurements of data' transfers. To enable not only external reconfiguration but internal reconfiguration, an ICAP (Internal Configuration Access Port) wrapped into the HWICAP IP is instantiated. The full exo-reconfiguration at reset is done through the external JTAG port. It is also optional to add an additional DMA to the Ethernet controller (thanks to the configuration of the IP in EDK) such as ”fast copy” incoming packets to dedicated buffer is possible. However and as lwIP port is not fully supporting DMA transactions, we left the controller without it.

2) Hardware DPR Protocol: We now explain the dynamic partial reconfiguration protocol in depth. A packet reception is divided into four steps (Figure 8). It goes through internal Ethernet FIFO to the reconfigurable area:

1) First, an Rx interruption occurrs. Received packet is stored into an internal Ethernet Controller FIFO.
2) Second, FIFO buffer is copied to RAM where memory pool (circular buffer) allocated by lwIP is available.
3) Third, Memory pool content is transferred to ICAP BRAM.
4) Finally, ICAP BRAM is written to the reconfigurable area.

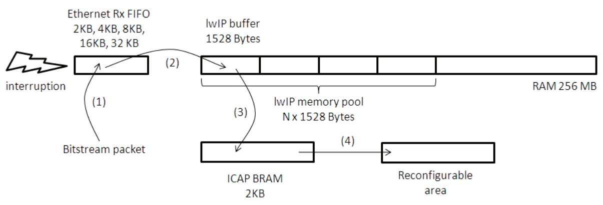

Fig. 8. Bitstream path from Ethernet to ICAP

Actually, more than 90% of the processing time is spent in data transfers from Ethernet controller buffer to lwIP memory pool and from memory pool to ICAP, thus considered to be the bottleneck. To ensure that our software/hardware partionning was the best, some evaluations were done based on four architectural options :

1) With processor data cache disable.
2) With processor data cache enable.
3) With processor data cache disable and DMA used to transfer memory pool content to ICAP BRAM.
4) With processor data cache enable and DMA for transfering.

In all these case, PPC405 instruction cache was activated and set to 16 KB large (8 BRAMs). When enabled the data cache is also 16 KB large. Table III demonstrates that software data copy with data cache enable is the best setup. This can be explained because EDK's DMA engine has no internal buffering, and doesn't do burst transfers. For processor without instruction cache, it might make sense to add a DMA, otherwise the inner loop of the optimized memory copy would be in cache and be executed at 2 cycles per instruction. The limiting factor will become the OPB latency (reading/writing from/to the OPB RAM). Indeed, when a data cache is enabled and as the processor exhibits cache coherency anomalies, it has to remain clean. It is the responsibility of the developer of ensuring that any buffers in cache which are passed to the DMA are flushed from it. In addition the cache has to be invalidated prior to using any buffers resulting of a DMA operation.

That is why we have decided to rely on pure software concerning all data transfers and mainly between lwIP memory pool buffers and ICAP BRAM. Moreover, this saves some hardware resources and decreases significant overhead due to the OPB to PLB bridge as well as avoiding additional managements by the PPC.

	1	2	3	4
throughput	-	+	-	+
hardware memory footprint	+	+	-	-
software memory footprint	+	+	-	-
overall	++	+++	-	+

TABLE III
HARDWARE/SOFTWARE PARTIONNING OPTIONS RESULTS

V. RESULTS

Our measures are based onto reconfiguration of PPC405 coprocessors in slave mode. Operators like multiplier, divider, but also more complex such as FFT and DCT were considered, producing partial bitstreams file size ranged from 60 KB to 200 KB. As our goal is to get the maximum throughput possible, this parameter was automatically setup with our bandwidth test previously studied. One can established that the optimum packet size given is always close to the Maximum Ethernet Transmission Unit (MTU), 1500 Bytes. Burst size was set to the maximum, be the total number of frames needed to send one bitstream depending on it's size. The Ethernet controller FIFO was fixed to 8 KB. The server was running the application protocol in slave mode so it initiating the transfer to the FPGA. Server Hardware is based onto an Asus Eee PC 1000H equipped with an Intel Atom 1.6 GHz, 1GB DDR2 RAM, Window XP 32bit home edition. Networking services were ensured by an Artheros AR8121/AR8113/AR8114 PCI-E Fast Ethernet Controller and an 802.11g Wireless LAN Card. Figure 10 sums up throughputs results as a function of the packet size and according to the scenarios given in Figure 9. These are now described in details.

A. LAN Topology Results

We have first envisaged a LAN configuration to ensure the global effectiveness of our protocol. It linked directly server and client machine via a simple crossover cable. Results obtained depend as we could expect, on transmitted packets size. We have obtained a sustainable 60 Mb/s throughput with an average packet size of 1492 bytes. This high transfer throughput matches with WIFI WLAN rate where ”only” 30 Mb/s is reachable. This first result has to be compared to related work: the endo-reconfiguration speed we have reached with our protocol is 15 times more efficient.

B. WLAN Topology Results

The second scenario describes the server which was connected to the client with a wireless link. FPGA was connected to a Ethernet/WIFI bridge removing the need of specific drivers. WIFI network type was set to ad-hoc allowing two or more wireless clients to be connected each other without any central controller. In this context, a constant 30 Mb/s throughput was reachable with the same LAN scenario software memory usage. This is the maximum physically achievable as the 54Mb/s possible by the 802.11g standard is pure theory.

To our best knowledge, no other works have proposed such a deal with dynamic partial reconfiguration.

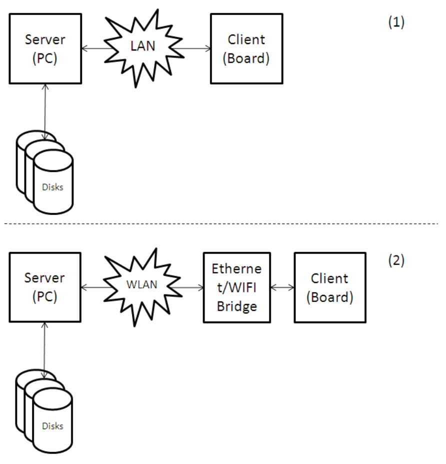

Fig. 9. LAN and WLAN protocol performance rating scenarios

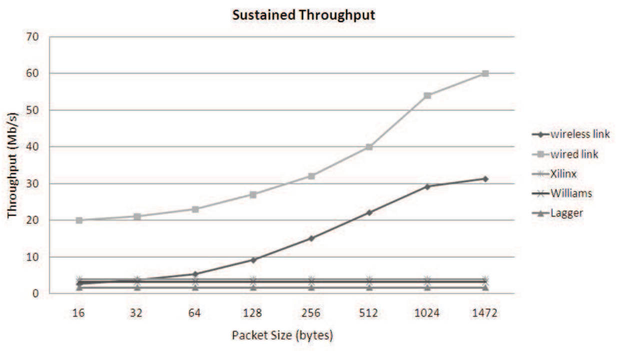

Fig. 10. Throughput curves for LAN and WLAN

C. Software and Hardware Memory Footprint

In terms of software, 100 KB are dedicated to executable code memory and 100 KB are allocated for data memory. This is 4 times less software memory compared to known previous works. In terms of hardware, all software instruction and data codes fits into BRAMs and then don't need any additionnal DDR RAM controller. Thus only 8% of the chip, 2524 over 32383 slices, is occupied. In a Virtex V2p a slice consists of two 4-input look-up tables and two flip-flops and is nearly equivalent to two Logic Element (LE) in Altera devices. This size is sufficient to implement user cicuits with the protocol on one FPGA. This brings our entire design to a lightweight implementation.

VI. Conclusion and Perspectives

Never taken into consideration in previous works, our PR platform takes advantage of wireless technology with high 30 Mb/s throughput which is the physical maximum achievable. Software memory cost is 200 KB, and only 8% of the chip is needed. Morevover, when used in a LAN topology, our implementation exhibits an order of magnitude gain (x15) compared to the best previous work, be 60 Mb/s. The underlying protocol is also simple and could be reused ”as is” thus, adding another session for doing one or another task is easy. From here we target other implementations and optimizations for reconfigurable embedded systems. First, IP addresses are assigned statically. We planned to do it more automatically and dynamically by implementing a DHCP (Dynamic Host Configuration Protocol) on the server depending of the context. This can also feed the fact that multiple devices are looking to dialog between each other as we necessary need a unique IP for identifying them. This can allow them to be added to a network without manual intervention. Next, when targeting systems without PPC405 hard cores, it might be welcome to port the application to a synthesizable Microblaze soft core at a cost of significant slices loss unfortunately. Moreover, it is worth noting that there is no security guarantee when exchanging data between the server and the client. Confidentiality, data integrity, authenticity (secure transaction in a word) are not addressed here, but this is anyway a good path to follow. In addition, even if not a standard, RUDP (Reliable UDP) should be investigated for being a protocol vastly employed for real time performances. As Virtex V2p is now considered as an efficient but sometimes deprecated part due to tools incompatibility, migrating to a Virtex V5 or even V6 in the near future will become a must be. Theses should deliver smaller partial bitstreams, thus smaller reconfiguration times, as well as a reduction of FPGA slices consumption. Last and not least when talking about network, Quality of Service (QoS) is essential for both LAN and WLAN. Number of works [GD04], [RAK] includes an additional software or hardware reliability layer over UDP to ensure correct data by implementing some simple algorithms. It could be a great idea to focus onto such methods where hostile environments could lead to packet looses.

References

[BCY+09] Pierre Bomel, Jeremie Crenne, Linfeng Ye, Jean-Philippe Diguet, and Guy Gogniat. Ultra-fast downloading of partial bitstreams through ethernet. In Mladen Berekovic, Christian Müller-Schloer, Christian Hochberger, and Stephan Wong, editors, *ARCS*, volume 5455 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2009.

[BDGC08] P. Bomel, J.-P. Diguet, G. Gogniat, and J. Crenne. Bitstreams repository hierarchy for fpga partially reconfigurable systems. *Parallel and Distributed Computing, 2008. ISPDC '08. International Symposium on*, pages 228–234, July 2008.

[CH02] Katherine Compton and Scott Hauck. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, June 2002.

[CZMS07] C. Claus, J. Zeppenfeld, F. Muller, and W. Stechele. Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system. *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07*, pages 1–6, April 2007.

[DGRB04] Jean Philippe Delahaye, Guy Gogniat, Christian Roland, and Pierre Bomel. software radio and dynamic reconfiguration on a DSP/FPGA platform. 2004.

[Dun01] Adam Dunkels. lwip. *Computer and Networks Architectures (CNA), Swedish Institute of Computer Science, http://www.sics.se/ãdam/lwip/*, 2001.

[Eth] Ethernet. Carrier sense multiple access with collision detection (csma/cd) access method and physical layer. *IEEE Standard 802.3*.

[GD04] Jasminder Grewal and John M. DeDourek. Provision of qoS in wireless networks. In *CNSR*, pages 337–340. IEEE Computer Society, 2004.

[Ins09] National Instruments. Building networked applications with the labwindows /cvi udp support library. January 2009.

[LUSG06] A. Lagger, A. Upegui, E. Sanchez, and I. Gonzalez. Self-reconfigurable pervasive platform for cryptographic application. *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pages 1–4, Aug. 2006.

[PA03] N.J. Ploplys and A.G. Alleyne. Udp network communications for distributed wireless control. *American Control Conference, 2003. Proceedings of the 2003*, 4:3335–3340 vol.4, June 2003.

[RAK] RAKNET. Raknet www.jenkinssoftware.com.

[RFC80] Internet Engineering Task Force. RFC768. User datagram protocol. August 1980.

[RFC81] Internet Engineering Task Force. RFC793. Transmission control protocol. September 1981.

[RFC89] Internet Engineering Task Force. RFC1122. Requirements for internet hosts – communication layers. October 1989.

[RSQ06] A.R. Rind, K. Shahzad, and M.A. Qadir. Evaluation and comparison of tcp and udp over wired-cum-wireless lan. *Multitopic Conference, 2006. INMIC '06. IEEE*, pages 337–342, Dec. 2006.

[Uch07] T. Uchida. Hardware-based tcp processor for gigabit ethernet. *Nuclear Science Symposium Conference Record, 2007. NSS '07. IEEE*, 1:309–315, 26 2007-Nov. 3 2007.

[WB04] John W. Williams and Neil Bergmann. Embedded linux as a platform for dynamically self-reconfiguring systems-on-chip. In Toomas P. Plaks, editor, *ERSA*, pages 163–169. CSREA Press, 2004.

[WIF] WIFI. Wireless lan medium access control (mac) and physical layer (phy) specifications.

[Xil06a] Xilinx. http://forums.xilinx.com/xlnx/attachments/xlnx/elinux/494/1/opb_hwicap.pdf. July 2006.

[Xil06b] Xilinx. Xapp433. web server design using microblaze soft processor. October 2006.