A Fast Prototyping Workflow for Reconfigurable SDR Applications

Alejandro. G. Gener
Networks and Embedded Systems
UTRC-Ireland
Cork, Ireland
garciaal@utrc.utc.com

Juan Valverde
Networks and Embedded Systems
UTRC-Ireland
Cork, Ireland
ValverJ@utrc.utc.com

J. Andrés Otero
Centro de Electronica Industrial
Universidad Politecnica de Madrid
Madrid, Spain
joseandres.otero@upm.es

Philip J. Harris
Networks and Embedded Systems
UTRC-Ireland
Cork, Ireland
HarrisPJ@utrc.utc.com

Abstract— **Nowadays, the level of complexity attained by embedded systems is convoluting the barrier between simulation and implementation. Dealing with complexity requires of new abstraction layers amongst design phases to guide the process from requirements to implementation. Model-based design methodologies offers an effective alternative to address these designs, but existing commercial tools are limited as new implementation technologies appear.**

This paper addresses this design problem by proposing an architecture and a methodology for fast prototyping of runtime adaptive Software Defined Radio applications on FPGAs. The methodology follows a model-based design approach including hardware-in-the-loop testing using automatic code generation. The processing architecture has been designed so Dynamic Partial Reconfiguration is possible to switch amongst different processing elements seamlessly at runtime. This approach speeds up the response for test iterations in SDR embedded designs going from hours to ten minutes, which is crucial to save costs.

Keywords—Model-based design, SDR, FPGA, DPR

I. Introduction

The design of complex embedded systems includes several stages ranging from requirements specification to hardware implementation. Model-Based Design (MBD) proves to be an effective and efficient way to trace the whole process, helping to overcome difficulties that might appear through visual inspection, simulation of models, easier fault injection or formal verification. In traditional design processes, information is usually transferred in the form of handwritten text documents, such as source code or batch files which can be difficult to understand by engineers without the specific background, while they are very time consuming and difficult to maintain and update. MBD tools, such as Matlab, allow engineers to focus on higher levels of abstraction and to take advantage of high levels of automation and automatic artefact generation to support certification [1].

This design approach is very powerful when dealing with safety critical applications such as space applications [2], commercial vehicle electronics [3], or aircraft avionics systems [4][5] where a complete traceability is a must and fast iterations in the design process help to overcome design problems in a faster and more controlled way.

One application that can take advantage of MBD techniques for hardware-software co-design is Software Defined Radio (SDR). For instance, The Wireless Avionics Intra-Communication (WAIC) association is enhancing wireless connectivity in avionics thanks to the use of Software Defined Radio (SDR) systems. In this context, following a MBD methodology accelerates the development process and reduces costs in the project without renouncing reliability [6].

Moreover, commercial Systems on Chip (SoCs) are becoming the preferred option to perform digital signal processing in SDR due their heterogeneous nature. They very often include general purpose processors, dedicated peripherals and programmable logic, typically in the form of FPGAs. Some of these FPGAs have the ability of accessing their own configuration memory and change their content at runtime without interrupting the execution of those parts left intact. This is called Dynamic Partial Reconfiguration (DPR) [7].

However, in spite of the fact that DPR meets SDR requirements with a better performance in terms of power consumption and resources [8], it is not included in fast prototyping MBD commercial flows. This enforces developers to combine these technologies manually, which is translated in an increased complexity, cost and iteration time for embedded tests.

This paper presents a new hardware-software architecture developed in a reconfigurable SoC that allows fast prototyping for SDR applications, reducing dramatically the time between testing iterations. In addition, Matlab MBD workflow has been extended to support DPR, providing a GUI which enables the application of the proposed methodology in real industrial applications. By using the integrated DPR feature, different types of signal processing elements such as modulators, equalizers or filters can be loaded seamlessly to enable new functionalities while increasing safety and security by adding on demand redundancies if needed. In order to ease the integration of the dynamic function blocks, the whole process is automated and guided using a Graphical User Interface (GUI). With this approach, any reconfigurable processing element can be developed and simulated at a high level of abstraction, enabling a fast track to generate a partial bitstream corresponding to the functionality under test and to introduce it into the embedded system for hardware-in-the-loop testing. Moreover, reconfiguration times can be measured in the hardware setup and introduced afterwards in subsequent simulations.

The rest of the paper is organized as follows. Section 2 contains the state of the art related with SDR platforms and MBD tools. Section 3 describes the baseline SDR development platform while Section 4 defines the modifications introduced in the architecture to make it compatible with DPR technology. In Section 5, dynamic partial reconfiguration workflow and their design specifications to achieve it are presented at the same time it describes the process automation and the GUI development. A real SDR application development and results are presented in Section 6 and Section 7 contains the conclusion and future work.

II. Related Work

A Software Defined Radio (SDR) is a programmable communication system with the capability of operating different wireless communication protocols without the need to change or update hardware components. It consists of a digital processing element, which executes signal processing operations such as modulations, Fast-Fourier Transfer (FFT functions), control gain, decoding or equalization and high-level protocol algorithms, connected to a transceiver, which modifies the signal to adapt their electromagnetic transmission characteristics. Digital signal processing algorithms can be performed by a general purpose processor (GPP), a digital signal processor (DSP), an FPGA or a SoC with a hybrid design (hardware-software architecture) [9].

SDR is not a new concept. It is possible to find a lot of research works optimizing different SDR processing elements such as GPPs [10], GPUs[11] and SoCs [12], being the latter the best solution due to its configurability and programmability [13].

Moreover, there are many different development boards, custom and commercial, in the state of the art for SDR applications: USRP devices [14] offers an expensive but solid FPGA-based SDR compatible with Matlab MDB tools meanwhile Xilinx development boards such as the Zedboard mixed with Analog Devices transceivers are a cheaper solution for Hybrid SDR systems [15]. Other companies such as Nvidia opts for GPU-based SDR systems offering a communication extension for the Jetson development board [16]. On the other hand, there are cheaper SDR solutions developed to stream data to host computers. The RTL-SDR transceiver [17] is the cheapest solution to stream data meanwhile the Beagleboard system [18] and the Adalm Pluto boards offer the possibility to process standalone complex algorithms without renouncing their low price. As a result of this analysis, the latter board has been selected for prototyping purposes in this work due to its cost and reconfigurability [19].

Despite mostly of the boards mentioned above are compatible with Matlab and GNU radio MBD solutions for embedded systems, these tools cannot generate embedded automatic code compatible with DPR. Current SDR solutions that implements DPR uses these tools to generate HDL code to modify it by hand, consuming hours in the implementation process [20].

On the other hand, some workflows based on the MBD approach have been developed introducing a DPR process [21] but none of them were specifically created for SDR architectures.

III. Baseline SDR Architecture

The Adalm Pluto board is a low-cost hybrid SDR system for radio applications. It combines a Zynq-7010 SoC connected to an AD9363 Analog Devices (AD) transceiver. It offers two channels for data transmission and reception that can be operated in full duplex, capable of generating or measuring RF analogue signals from 325 to 3800 MHz, at up to 61.44 Mega Samples per Second (MSPS) with a 20 MHz bandwidth. The board is a completely self-contained device entirely USB powered with the default firmware that offers an interaction with RF signals from MATLAB, Simulink, GNU Radio or custom C, C++, C#, or Python. The firmware and hardware specifications are fully documented by AD offering a complete system customization.

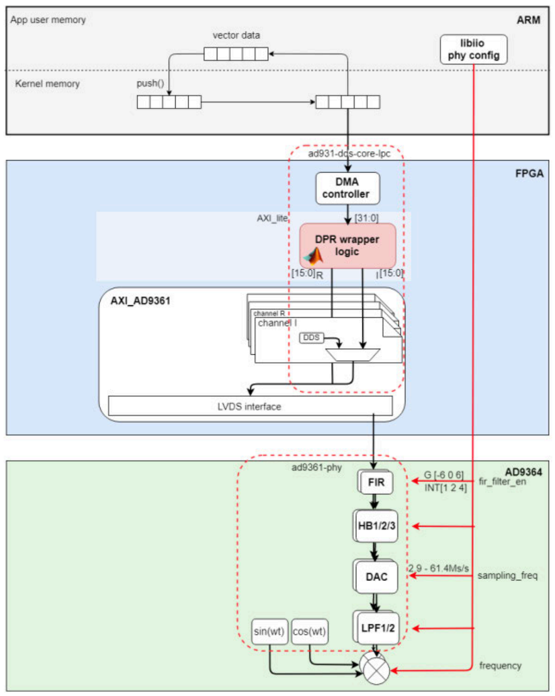


Figure 1. DPR logic block implemented inside the SDR architecture

The SoC is featured with a default Embedded Linux Operating System distribution with a Buildroot file system which contains pre-compiled libraries to access the AD transceiver. Data are stored in buffers that are pushed into the kernel memory space to be collected by the Programmable logic (PL) part of the SoC through a Direct Memory Access (DMA) data mover. The DMA sends this data to the interpolator that can be configured from software using the provided AD libraries. This interpolator can be enabled providing an up-sampling factor of eight and dividing data in two different channels for RF transmission (I/Q) to be managed by the AXI_AD9361 HDL block [22]. This processing element manages the communication with the AD9364 transceiver through an LVDS protocol. It also generates a clock signal using an internal phase-locked loop (PLL) that supplies the interpolator and direct memory access (DMA) logic. This signal depends directly on the digital-to-analogue converter (DAC) sampling frequency providing a complete synchronization amongst elements (DMA, interpolator and DAC).

The first element that the AD9364 transceiver has is a programmable FIR filter whose parameters (such as coefficients, gain and interpolator factor) can be configured from software using a SPI communication. Then, data are filtered before and after entering in the DAC. Finally, the analogue signal is mixed with the carrier to be transmitted through the antenna. All the different communication parameters (carrier frequency, sampling frequency, filters and bandwidth) can be easily defined and modified from the software side.

This default architecture is designed to stream radio-frequency data to a host computer connected via USB. Some tools such as Matlab or GNU radio are able to communicate with the board and run simulations using real RF signals.

IV. Modified SDR Architecture

After the description of the baseline reference architecture for SDR, the modifications introduced to the architecture in order to make it compatible with dynamic and partial reconfiguration are described next.

First of all, the interpolator logic has been replaced by a hardware block that wraps and manages the different functionalities that can be dynamically added or modified to the baseline model. Figure 1 shows how this idea fits with the original structure. The purpose of the dynamic architecture is to be able to change between two different paths at run time reconfiguring the specific hardware logic of the system seamlessly.

However, from the software point of view, the user does not have knowledge about which precise bit is received by the DPR wrapper logic block. For this reason, the implementation of a protocol is necessary to guaranty a perfect hardware-software synchronization.

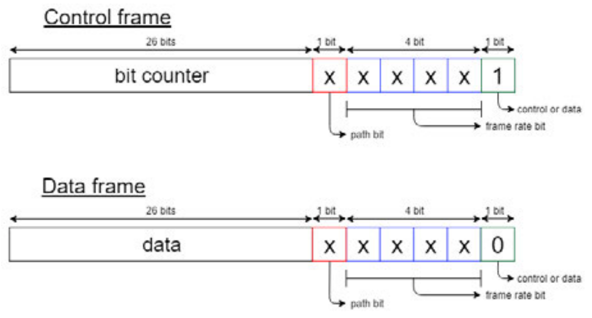


Figure 2. Protocol for hardware-software synchronization

DMA frames are divided in two groups as Figure 2 shows: control frames and data frames. The first ones contain the information about how many bits are going to be sent through a specific path. In turn, data frames contain data bits to be sent. However, both frames have a 6-bit header for synchronization purpose. The first bit indicates the frame type (data or control), the next four bits select one of twelve different oversampling factors, and the last header bit indicates which path has to be enabled. This protocol can be easily adapted if more reconfigurable blocks or configuration parameters need to be used.

The first frame read by the system has to be a control frame while next frames contain the data to be transmitted. Once the data length indicated in the control frame has been delivered, another control frame is sent indicating how many bits would be delivered with that specific configuration. This allows the architecture to change at run time the processing path without losing a single bit of information. Moreover, the fix 6-bit header creates a reliable capability in case of any desynchronization between software and hardware appears.

The proposed DPR wrapper logic is composed of four main elements: A control state machine logic, a sample rate

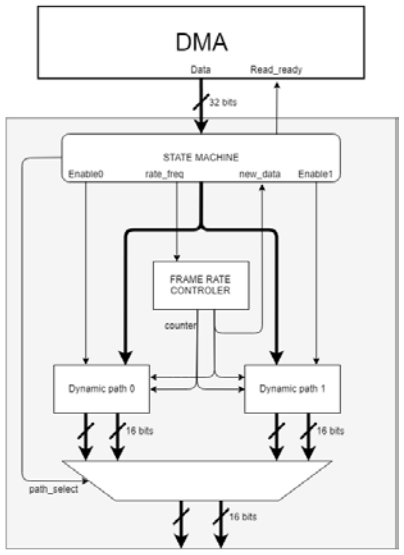


Figure 3. DPR wrapper logic block scheme

controller, two parallel dynamically reconfigurable blocks and a multiplexer. Figure 2 shows the connections amongst DPR wrapper elements and its interaction with the DMA.

A. *Control State Machine Logic*

The state machine manages the interface between the DMA and the data decoder. The DMA stores the data sent from software in 32-bit frames which are read from a FIFO. The wrapper logic reads the last data and sends a ready signal to the DMA indicating that the frame has been read. Once the frame is read, it has to be decoded.

B. *Sample Rate Controller*

The frame rate controller block receives the up-sampling factor information from the control state machine. It will generate a counter with a different number of steps from 0 to 1280 depending on the up-sampling factor selected by the user. This counter supplies both reconfigurable paths that can use it for many DSP algorithms such as modulations, interpolations or filtering. This block also indicates to the state machine when a new information bit can be sent to the different paths.

C. *Dynamically Reconfigurable Blocks*

These blocks represent the different reconfigurable modules that can be updated to the model at run-time. From the static architecture point of view, they are black boxes that can be filled with any logic algorithm, whenever it follows the rules mentioned below:

- The algorithm has to work with a throughput of one sample per clock cycle, to maintain synchronism in the data path.
- It has to divide the signal in two sixteen bit integers for the different electromagnetic waves (real and imaginary).
- It will receive data from the state machine sequentially, bit by bit.

D. *Reconfigurable Path Multiplexer*

The multiplexer is controlled by the state machine which reads the path selection bit from the control frame According to this bit, the configured path is selected to send its output bits to the AXI_AD9361 HDL module. In the meanwhile, the unselected path can be reconfigured.

One of the main advantages of this architecture is not only that changes can happen at runtime without losing any information bit but also a huge sampling frequency drop appears (eq 1.) between the DAC frequency and the software, allowing the microprocessor to have more time between each time the microprocessor fills the buffer with new data.

Being F_{DAC} the DAC sampling frequency, T_{over} the over-sampling factor that indicates the number of signal samples per bit, DMA_{width} the number of data bit per DMA frame and $BUFFER_{width}$ the number of 32-bit frames that the software buffer has, the sampling frequency from the software point of view can be described as:

$$Freq_{sw} = \frac{F_{DAC}}{T_{over} * DMA_{width} * BUFFER_{width}} \qquad (1)$$

For instance, if the DAC is configured with 61.4 MSPS (which is the maximum physical sample rate available for this model), with an oversampling factor of 1280 (maximum configurable for the architecture) and a buffer of 4096 frames. The period of time that the processor has between buffer pushes is:

$$Period = \frac{1280 * 26 * 4096}{61.4 * 10^6} \approx 2.23\ s \qquad (2)$$

Note that DMA_{width} is 26 because header bits do not count as data bits.

This sampling frequency drop gives the microprocessor the possibility to perform more complex algorithms between buffer fill operations such as Fast Fourier transforms (FFTs) or, like in the application presented in this paper, dynamic partial reconfiguration.

TABLE I. RECONFIGURABLE SECTION RESOURCES

Region	Resources				
	LUTs	***Registers***	***Muxes***	***BRAMs***	***DSPs***
Pblock0	2000	4000	1000	10	10
Pblock1	4000	8800	2200	20	20
ZYNQ 7010	17000	35200	16000	60	80

V. DPR WORKFLOW AND GUI

Dynamic Partial Reconfiguration can be defined as the modification of an operating FPGA design by loading a partial configuration file at runtime, while the area of the device is not affected by the reconfiguration remains working [23]. This allows FPGAs to carry out different functionalities without penalizing its size reusing a specific area each time a change in the functionality is needed. Vivado partial reconfiguration workflow eases the generation of partial bitstreams compatible with previously synthesized systems. A static part (i.e. the portion of the design not affected by reconfiguration) is previously synthesized with dummy boxes instead of the partial reconfiguration modules (PRMs). Before the placement and routing operations, it is necessary to define which regions of the FPGA will allocate the different PRMs. This has to be done manually since it depends on the developer criteria. Once the placement and routing are done, all the logic that belongs to the static part is locked and saved in a checkpoint that contains the information of the static architecture. It can then be used for the implementation of reconfigurable blocks [24].

The implementation of the static architecture explained in section II featured with dynamic partial reconfiguration cannot be completed using Matlab embedded hardware-software co-design workflow since this feature is not fully supported by this workflow. Nevertheless, the architecture has been designed in Simulink to test its functionality before the implementation. Then, the VHDL code of the static model is

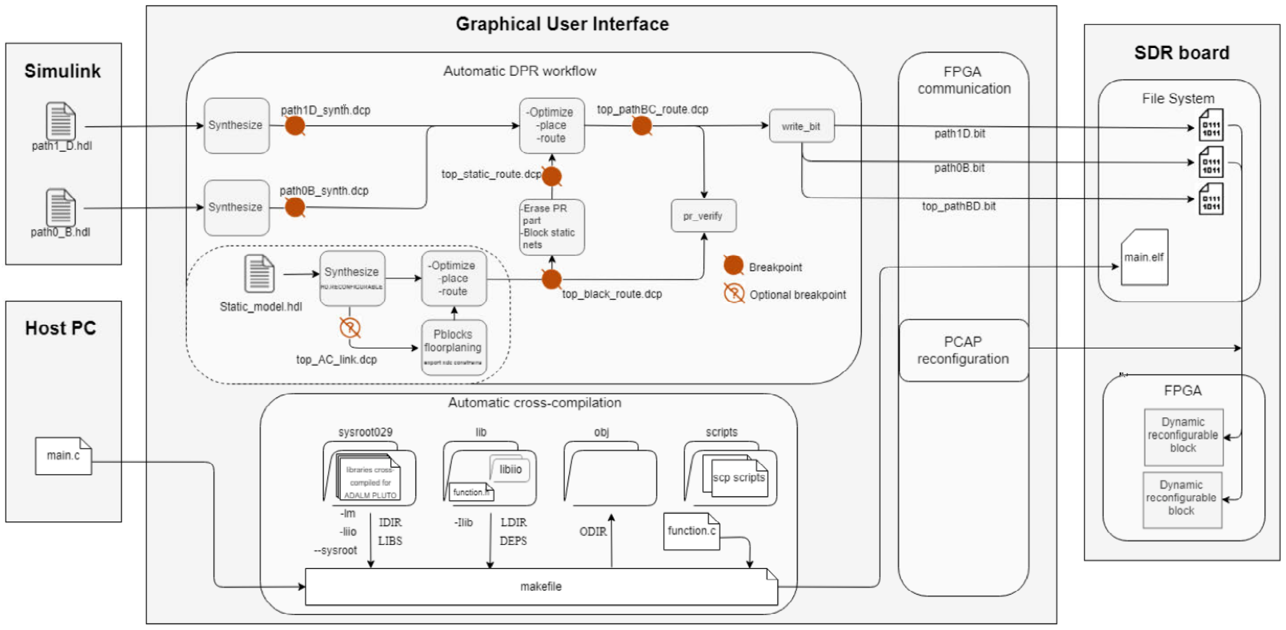


Figure 4. Dynamic Partial Reconfiguration workflow.

generated automatically using Matlab HDL coder [25] to introduce it as a custom IP inside the baseline board design. Following Vivado DPR workflow, the reconfigurable paths are implemented in the model introducing black boxes that bypass data directly to the AXI_AD9361 HDL module. In table I it is shown the amount of resources dedicated to each reconfigurable region of the FPGA. It is important to notice that the difference between reconfigurable region sizes allows developers to allocate algorithms inside one region or another depending on the resources they need. Finally, a Linux image has to be reconstructed with the bitstream generated to link software resources with the new hardware implementation.

Once the bitstream of the static architecture is generated (and even running), new reconfiguration modules can be added to the system following the DPR workflow. One of the main advantages of Vivado tools and Partial Reconfiguration workflow is that it can be automated thank to the use of TCL scripts. Taking advantage of TCL commands, a Graphical User Interface (GUI) has been developed in Phyton to enhance the implementation speed of dynamic partial reconfiguration blocks from Simulink models. The process of building a standalone application using custom hardware accelerators designed from Simulink can be divided in 4 main tasks: HDL code generation, partial bitstream generation, FPGA programming and cross-compilation. As it is shown in Figure 4, the GUI manages 3 of the tasks automatically since the HDL code generation is operated from Simulink.

A. HDL code generation

The first step refers to the VHDL code generation from Simulink. As it was mentioned before, DPR feature is not directly compatible with HDL coder. However, the GUI can merge VHDL codes generated from Simulink with the static design. To do so, the simulation model in Simulink must comply with the following rules:

- The name of the PRM block has to be the same as the black box allocated in the static Vivado model.
- The interfaces between PRMs and the static model must match in terms of names and data types.
- The algorithm block must be wrapped by a Simulink subsystem with registers in each Input and Output signals to ensure timing FPGA specifications.

B. DPR Workflow

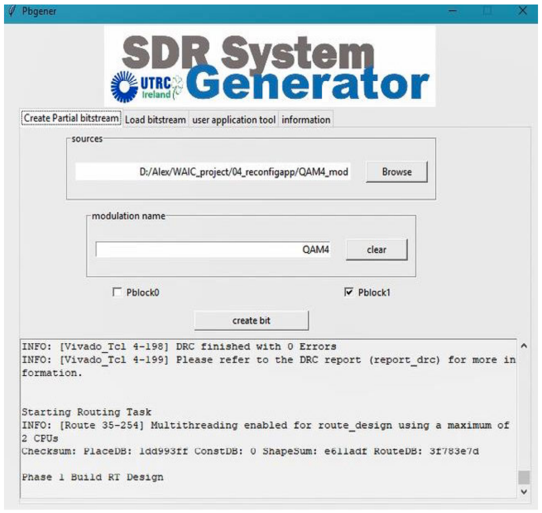
Once the HDL code is generated, the reconfigurable block is synthesized separately to be merged with the previously generated static design. This block is mapped into the regions of the FPGA selected as reconfigurable areas during the implementation of the static system and, after the verification process, the partial bitstream is created. This process, as it is shown in Figure 4, is managed automatically by the GUI, which calls a TCL script in Vivado that starts the DPR workflow in a second plane with the configuration specified in the GUI such as the name of the partial bitstream and the FPGA region that will allocate the PRM block. Moreover, all the progress can be followed at run-time through the GUI command window.

C. Send bitstreams and program FPGA

The partial bitstream is sent to the board file system using a Session Control Protocol (SCP) between the board and the host computer, also integrated in the GUI. The GUI is prepared to open a SSH session (password and username are added automatically) to communicate itself directly with the embedded Linux. It can access directly to the Linux PCAP driver that allows the processor to access to the FPGA program memory in order to load the specific partial bitstream into the FPGA memory. Moreover, the GUI is prepared to bind and unbid the necessary drivers to upload not only a partial bitstream but also a total bitstream with a different model, enhancing the modularity of this tool for applications different than SDR [26].

D. Cross-compile applications

Once the hardware is uploaded to the board, it is necessary to create a software application that sends input data to be transmitted through the architecture mentioned before. This application has to be cross-compiled in the host computer since the board embedded Linux distribution does not support an embedded compiler. The GUI has stored not only AD library cross-compiled, but also an extra abstraction layer library to ease the transceiver and hardware configuration. The



sources: D:/Alex/WAIC_project/04_reconfigapp/QAM4_mod

modulation name: QAM4

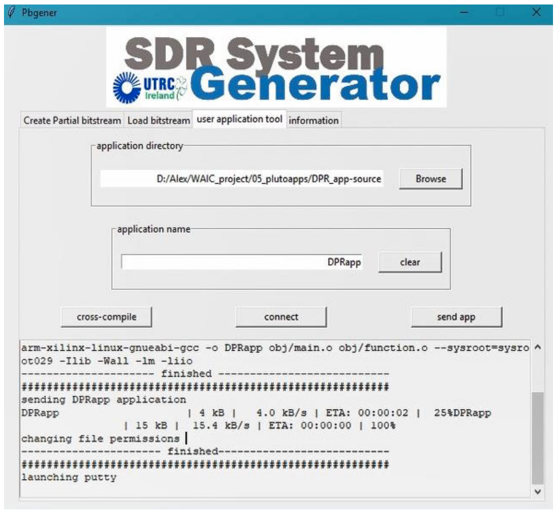
Pblock0 [✓ Pblock1]

```
INFO: [Vivado_Tcl 4-198] DRC finished with 0 Errors
INFO: [Vivado_Tcl 4-199] Please refer to the DRC report (report_drc) for more in
formation.


Starting Routing Task
INFO: [Route 35-254] Multithreading enabled for route_design using a maximum of
2 CPUs
Checksum: PlaceDB: 1dd993ff ConstDB: 0 ShapeSum: e611adf RouteDB: 3f783e7d

Phase 1 Build RT Design
```

(a) GUI Bitstream generator window



application directory: D:/Alex/WAIC_project/05_plutoapps/DPR_app-source

application name: DPRapp

```
arm-xilinx-linux-gnueabi-gcc -o DPRapp obj/main.o obj/function.o --sysroot=sysro
ot029 -Ilib -Wall -lm -liio
--------------------- finished --------------------------
#########################################################
sending DPRapp application
DPRapp                    | 4 kB |   4.0 kB/s | ETA: 00:00:02 |  25%DPRapp
                | 15 kB |  15.4 kB/s | ETA: 00:00:00 | 100%
changing file permissions |
---------------------- finished--------------------------
#########################################################
launching putty
```

(b) GUI Cross-compiler window

Figure 5. GUI. Sub-figure (a) shows the bitstream generator window and sub-figure (b) shows the cross-compiler window.

GUI is capable of generating an application that runs standalone inside the board for a fast hardware-in-the-loop testing. Run-time measures to Matlab are sent through an SSH connection.

Figure 5 shows 2 of the 4 GUI windows: The partial bitstream generator in a) and the cross-compiler window in b). They are composed by a command window that shows information of second plane processes at run-time, a browse to select the specific files needed for each task, a text input box to select the name and a set of buttons to launch the different workflow tasks. Moreover, the partial bitstream generator includes the selection of the FPGA region where the PRM will be allocated. The bitstream loader window has a similar structure to the partial bitstream generator and the information window offers instructions about how to use the application and information about the resources of each reconfigurable FPGA region.

In order to speed up the testing of reconfigurable blocks, two software applications had been introduced by default in the board file system. These applications send constantly random input data through one of the two paths.

VI. SDR TEST APPLICATION

Thanks to Matlab automatic code generation and the GUI mentioned before, it is possible to develop a SDR application which runs standalone in the board compatible with DPR going from simulation to real implementation in a few minutes. To demonstrate this approach, an adaptive modulation application has been created using the workflow described above.

Adaptive modulation is an approach in cognitive radios [27] that has being used in multiple communication applications such as GSM (Global System for Mobile Communication). It uses an adaptive architecture to modify the modulation of the physical layer according to the different weather conditions since there are certain modulations that are more robust under raining conditions.

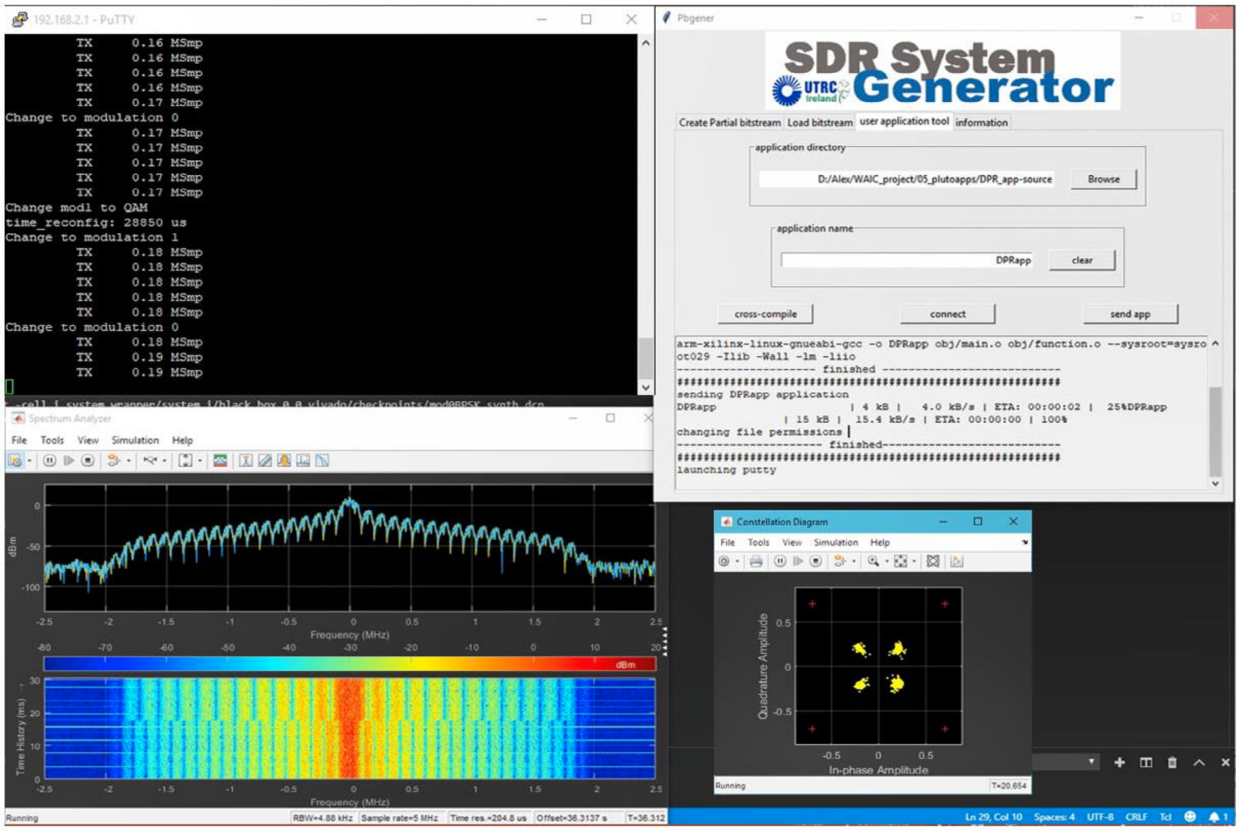
To develop the use case application, the Simulink model created for testing the static architecture has been merged with three different modulation blocks, also designed in Simulink. These blocks contain three different modulation algorithms which will replace the dummy boxes implemented in the static architecture. Moreover, its functionality can be first simulated in Simulink to ensure the correct functionality of the algorithms. The modulations selected for these applications are GMSK, QPSK and 4QAM.

GMSK modulation signal $x(t)$ is transmitted following eq. 3 being $\Phi(t)$ the phase of the demodulated signal and f_c the carrier signal frequency.

$$x(t) = \cos\big(\Phi(t)\big)\cos(2\pi f_c t) - \sin\big(\Phi(t)\big)\sin(2\pi f_c t) \quad (3)$$

The modulator will transform data bits into positive and negative symbols and depending on the sign, it reads the elementary phase symbol data from look-up-tables [28]. The symbol is added with previous samples and divided into I/Q signals thanks to the proper operation (sin or cos) to obtain eq. 3. This last operation has been developed with a pipelined CORDIC architecture following the specification of having a throughput of one sample per clock cycle. [29]

On the other hand, 4QAM and QPSK modulations have an input buffer that packages the number of bits needed for each modulation (2 bits for QPSK and 4 bits for 4QAM). The proper constellation value is obtained from a Look-Up-Table according to the bits received. Then, the symbols pass through a raise cosine filter in order to reduce Inter-Symbol Interference (ISI) and are sent to the output.



```
         TX      0.16 MSmp
         TX      0.16 MSmp
         TX      0.16 MSmp
         TX      0.16 MSmp
         TX      0.17 MSmp
Change to modulation 0
         TX      0.17 MSmp
         TX      0.17 MSmp
         TX      0.17 MSmp
         TX      0.17 MSmp
         TX      0.17 MSmp
Change modl to QAM
time_reconfig: 28850 us
Change to modulation 1
         TX      0.18 MSmp
         TX      0.18 MSmp
         TX      0.18 MSmp
         TX      0.18 MSmp
         TX      0.18 MSmp
Change to modulation 0
         TX      0.18 MSmp
         TX      0.19 MSmp
         TX      0.19 MSmp
```

```
arm-xilinx-linux-gnueabi-gcc -o DPRapp obj/main.o obj/function.o --sysroot=sysro
ot029 -Ilib -Wall -lm -liio
--------------------- finished -----------------------------
############################################################
sending DPRapp application
DPRapp                    | 4 kB |   4.0 kB/s | ETA: 00:00:02 |  25%DPRapp
                | 15 kB |  15.4 kB/s | ETA: 00:00:00 | 100%
changing file permissions |
--------------------- finished------------------------------
############################################################
launching putty
```

Figure 6. DPR application running standalone. The figure shows a SSH session and data represented by Matlab in the frequency and complex domain.

Once the modulations have been tested in Simulink, the partial bitstreams are generated directly from the HDL files automatically generated by Matlab and sent them to the board. The testing of each modulation can be done using the two applications loaded in the file system that sends random data through both reconfigurable paths on-board. This allows a faster approach to test different signal processing algorithms designed independently in Simulink in a real environment.

A specific application has been developed in order to test dynamic partial reconfiguration at runtime. This program takes the partial bitstreams from the board file system and upload them into RAM memory to speed up the reconfiguration time. QPSK modulation will run into path0 dynamic block meanwhile GMSK and 4QAM are constantly reprogramed into path1 block. Having partial bitstreams loaded in the system, the application first sends random data through GMSK modulation, which is allocated in path1. Then, it changes to QPSK block (path0) meanwhile it is reconfiguring path1 with 4QAM modulation. Thanks to the benefits provided by dynamic and partial reconfiguration, the system can continue sending data through the other path without interrupting the communication. Once the system finishes to send data through path0, it will change to path1, modulating the signal with the new configured PRM.

For a hardware-in-the-loop testing, another board is connected to the host computer. While one board is transmitting data standalone with the application mentioned above, the other is streaming received data directly to Matlab. Thanks to this implementation it is possible to visualize the signal sent at runtime. Nevertheless, it is possible to test real measurements with only one board since it allows full-duplex communication. The architecture can run a standalone application to send data while it is streaming the received data to Matlab at the same time. Figure 6 presents the spectrum and the spectrogram of the received signal while modulations are changing. It also shows the reconfiguration time of 2.8 ms for each PRM. This time represents the reconfiguration of a quarter size of the FPGA through the PCAP driver and can be decreased using different reconfiguration processes such as the ICAP.

Thanks to this application it is proved that new reconfigurable signal processing algorithms such as different modulations for a SDR system can be implemented and tested in less than ten minutes while a complete implementation of the modulation would take hours or even days. Moreover, it is proved that the reconfiguration of dynamic blocks can be done at run-time without affecting the communication reliability giving at the same time information about partial reconfiguration times.

VII. Conclusions

This paper has presented not only a new hardware-software architecture developed in a SoC that allows a fast prototyping solution for SDR applications, but also a new MBD workflow compatible with DPR for software-hardware designs. The architecture, implemented in the ADALM PLUTO board, has been designed to allow engineers to add dynamic function blocks exchangeable in run-time thanks to DPR. Moreover, all the process is automated and presented in a visual way using a GUI. The process follows a MBD approach adding partial bitstream generation from VHDL code generated with Simulink. With this approach, designers can develop and simulate in Simulink any processing element, generate a partial bitstream and introduce it into the embedded system for a hardware-in-the-loop testing in less than ten minutes, saving developing time and costs in complex SDR projects. The partial reconfiguration times can be measured and introduced in future simulations to increase the similarity between them and real reconfigurable SDR systems.

Moreover, the GUI can also generate partial bitstreams for any static model and for any Zynq development board, such as Zedboard or Pynq. It has been proven to be an appropriate solution for a fast prototyping not only in SDR applications but also in any application that requires a SoC implementation. This approach also opens up a bigger customization than the supported by Matlab hardware-software co-design workflow, implementing DPR in embedded systems.

References

[1] F. Paternò, *Model-Based Design and Evaluation of Interactive Applications*, Springer, 1999

[2] P. Angeletti, M. Lisi, and P. Tognolatti, “Software Defined Radio: A key technology for flexibility and reconfigurability in space applications,” in 2014 IEEE Metrology for Aerospace (MetroAeroSpace), may 2014, pp. 399–403

[3] K. D. Singh, P. Rawat, and J.-M. Bonnin, “Cognitive radio for vehicular ad hoc networks (CR-VANETs): approaches and challenges,” EURASIP Journal on Wireless Communications and Networking, vol. 2014, no. 1, p. 49, dec 2014.

[4] T. Kazaz, C. Van Praet, M. Kulin, P. Willemen, and I. Moerman, “Hardware Accelerated SDR Platform for Adaptive Air Interfaces,” apr 2017.

[5] X. Cai, M. Zhou, and X. Huang, “Model-Based Design for Software Defined Radio on an FPGA,” IEEE Access, vol. 5, pp. 8276–8283, 2017.

[6] “WAIC Project” [Online]. Available: https://waic.avsi.aero/about/

[7] T. Kalb and D. Göhringer, “Enabling dynamic and partial reconfiguration in Xilinx SDSoC,” 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, 2016, pp. 1-7.

[8] S. Hosny, E. Elnader, M. Gamal, A. Hussien, A. H. Khalil and H. Mostafa, "A Software Defined Radio Transceiver Based on Dynamic Partial Reconfiguration," 2018 New Generation of CAS (NGCAS), Valletta, 2018, pp. 158-161.

[9] Akeela, R.; Dezfouli, B. Software-defined Radios: Architecture, state-of-the-art, and challenges. Comput. Commun. 2018, 128, 106–125.

[10] K. Tan, H. Liu, J. Zhang, Y. Zhang, J. Fang, and G. M. Voelker, “Sora: high-performance software radio using general-purpose multicore processors,” Communications of the ACM, vol. 54, no. 1, p. 99, jan 2011.

[11] J. G. Millage, “GPU Integration into a Software Defined Radio Framework,” Ph.D. dissertation, Iowa State University, 2010.

[12] J. van de Belt, P. D. Sutton, and L. E. Doyle, “Accelerating software radio: Iris on the Zynq SoC,” in 2013 IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC), oct 2013, pp. 294–295.

[13] B. Cope, P. Y. Cheung, W. Luk, and L. Howes, “Performance Comparison of Graphics Processors to Reconfigurable Logic: A Case Study,” IEEE Transactions on Computers, vol. 59, no. 4, pp. 433–448, apr 2010.

[14] “USRP devices for SDR” [Online]. Available: https://www.ettus.com /product-categories/usrp-networked-series/

[15] “Zynq boards for SDR” [Online]. Available: http://zedboard.org/product/zedboard-sdr-ii-evaluation-kit

[16] “Nvidia GPU for SDR” [Online]. Available: https://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html

[17] “RTL-SDR specifications” [Online]. Available: https://www.rtl-sdr.com/about-rtl-sdr/

[18] “beagleboard” [Online]. Available: http://beagleboard.org/project/Beagle+SDR/

[19] “Adalm Pluto SDR development board” [Online]. Available: https://wiki.analog.com/university/tools/pluto

[20] R. Torrego, I. Val, E. Muxika, "OQPSK Cognitive Modulator Fully Fpga-Implemented Via Dynamic Partial Reconfiguration And Rapid Prototyping Tools" in *Procedings of SDR'11-WlnnComm-Europe,* 22-24 Jun 2011.

[21] G. Ochoa-Ruiz, P. Wattebled, M. Touiza, F. De Lamotte, "A Modelling Front-End for Seamless Design and Generation of Context-Aware Dynamically Reconfigurable Systems-on-Chip" *Journal of Parallel and Distributed Computing,* February 2018, Pages 1-19.

[22] "AXI_AD9361 HDL block" [Online]. Available: https://wiki.analog.com/resources/fpga/docs/axi_ad9361

[23] L. Wang, F. Wu, "Dynamic Partial Reconfiguration in FPGAs" in *2009 Third International Symposium on Intelligent Information Technology Application* 21-22 Nov. 2009

[24] "Xilinx Dynamic partial reconfiguration workflow" [Online] Available:https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug909-vivado-partial-reconfiguration.pdf

[25] "HDL Coder." [Online]. Available: https://www.mathworks.com/products/hdl-coder.html

[26] M. Al kadi, P. Rudolph, D. Gohringer, M. Hubner, "Dynamic and partial reconfiguration of Zynq 7000 under Linux" in *International Conference on ReConFigurable Computing and FPGAs (ReConFig)* December 2013.

[27] Adel Ghazel Zhi, Youseff ''Otimized DSP implementation of GMK software Modem for GSM transceiver", 51L IEEE Vehicular Technology Conference Proceedings, Volume 3, pp 2513-2577, 2002.

[28] "GMSK modulator for DSP applications" [Online] Available: http://www.ti.com/lit/an/spra139/spra139.pdf

[29] M. A. El-Motaz et al., "A CORDIC-Friendly FFT Architecture", IEEE IWCMC, pp. 1087-1092, 2014