Reconfigurable Signed Multiplier (2C and SM) on Zynq-7000 PSoC

Hussein Alawsi, Andrew Meesseman
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: husseinalawsi@oakland.edu, ameesseman@oakland.edu

***Abstract*—This project demonstrates the utilization of dynamic partial reconfiguration on a Zynq-7000 All Programmable System-on-Chip to switch between a 2's complement and sign and magnitude multiplier. The PCAP interface is used to reconfigure the FPGA fabric on-the-fly to switch between these two number representations. The use of a hardware generated interrupt is implemented to control when to perform this partial reconfiguration. The AXI4-Full protocol is used to interface with a signed multiplication IP. The AXI4-Lite protocol is used to interface with a GPIO IP for determining which reconfigurable module to reconfigure the circuit with.**

I. Introduction

This report will cover the process that was developed for the experimentation and testing of this project. Vivado 2019.1 and the Xilinx SDK were used to write the VHDL and C source code that was used to control the processing system (PS) and FPGA fabric (PL). Results of the project will also be discussed in detail to understand future project viability. How these elements were created, simulated, and tested will be further discussed below. The purpose for creating this project is to be able to show the versatility of partial reconfiguration in an embedded system.

II. Methodology

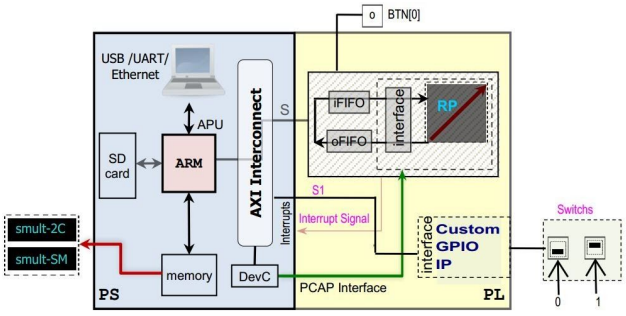

Fig. 1. High level PS and PL connection

The system consists of an AXI interconnect between the PS and PL for two custom AXI slave peripherals that were created. The first is an AXI4-Full peripheral, capable of performing signed multiplication and producing the product of two 8-bit inputs, which hosts the reconfigurable partition (RP) of the system. This RP can be programmed with a partial bitstream that allows either 2's complement (2C) or sign and magnitude (SM) multiplication. This IP also generates a PL-PS (PL to PS) interrupt using a button input (Button 0 on the Zybo Z7). This allows the PS to determine when to perform a dynamic partial reconfiguration (DPR) of the system. The second peripheral uses AXI4-Lite and is able to pass along two I/O values (such as Switch 0 and Switch 1 in this case) from the Zybo Z7 board to the PS when needed. In the system that has been created, these two I/O's are being used to determine which circuit to program to the RP during DPR.

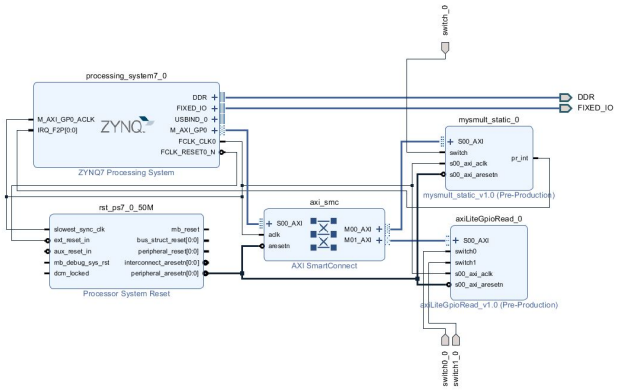

Fig. 2. High Level Static Block Design

Fig. 2 shows a high level block diagram of the components in the system, as created with the block design tool in Vivado. It can be noted that the custom AXI peripherals, the signed multiplier (mysmult_static) and the GPIO passthrough (axiLiteGpioRead), are connected to the PS via the AXI interconnect. This allows and handles the PS communication with both peripherals. The value "switch_0" is connected to a button (used to ultimately generate the PS-PL interrupt), and "switch0_0" and "switch1_0" are connected to switches of the Zybo board (as described in the project constraint file).

The process of reaching this system design will be outlined to show the steps that were taken to ensure system functionality. These steps were followed very precisely to ensure a working system and to minimize issues and delays.

A. *Design VHDL Circuits*

The first step in this project was generating all the multiplier circuits that were going to be used (unsigned multiplier, along with 2C logic and SM logic). There were three main portions: the main one was the unsigned multiplier as shown in Fig. 3 (which will be later on a part of the static portion of the system design). Keep in mind that this circuit is connected internally to the RP. The RP contains the other two circuits, the reconfigurable modules (RM), which are the 2's complement (2C) circuit (Fig. 5) and the sign and magnitude (SM) circuit (Fig. 6). Both circuits are responsible for checking the signs of the input numbers and deciding the sign of the final value, along with generating an absolute value of the inputs for the unsigned multiplier to be able to work with.

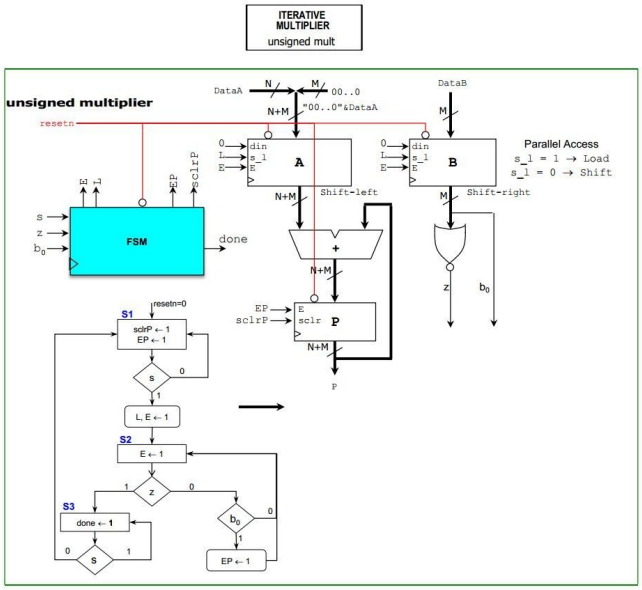

Fig. 3. Unsigned Multiplier Architecture

Fig. 3 shows the architecture of the unsigned multiplier used in the system. Both the 2C and SM circuits require the use of an unsigned multiplier in their architecture. As this is needed for both multiplication modes, this circuit is kept in the static portion of the circuit.

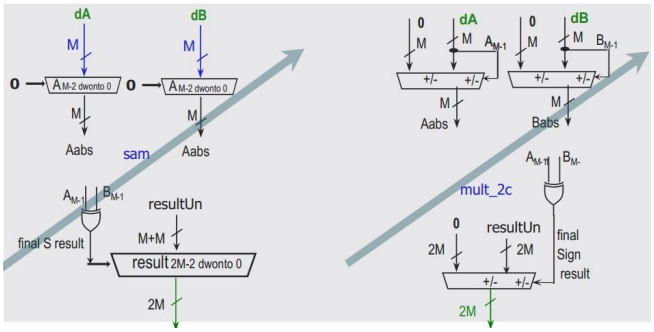

Fig. 4. Reconfigurable Modules (RM's)

Fig. 4 shows the logic that differentiates the 2C from the SM multipliers. Given that this is the only difference between the two multipliers, only these components are included in the RP of the system. The line through each of these circuits is included to show that these circuits are reconfigurable. Refer to Fig. 5 below to see their implementations.

After the design and testing of each circuit alone was completed, the circuits were then separated inside the system into either a static or reconfigurable part as seen fit. As stated above, the unsigned multiplier is a static component while the 2C and SM logic circuits are considered to be reconfigurable. A new circuit is designed to host either the logic for 2C and SM signed multiplication (based on a parameter in the file) because of their reconfigurable nature. This allows for easy circuit synthesis and mitigates problems when generating the partial bitstreams needed for the DPR.

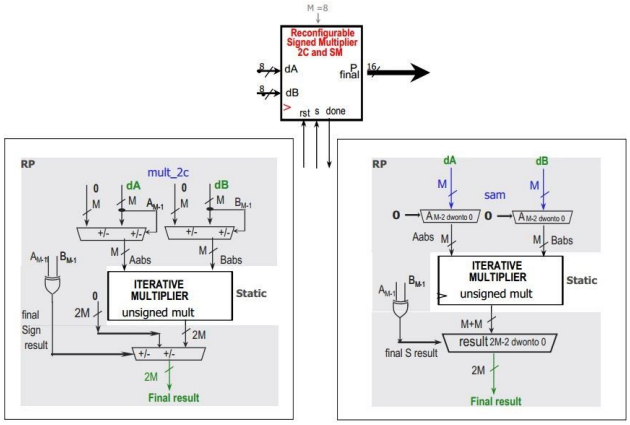

Fig. 5. Static and RP Implementation for Multiplier

Fig. 5 shows both the implementation of the 2C (left) and SM (right) circuits. It may be noted that the gray portion in each circuit is partially reconfigurable and is configured with either the 2C or SM components as seen in Fig. 4.

The 2C circuit is the first RM. It takes the absolute values of the inputs and sends them to the unsigned multiplier (in the static portion of the circuit). It then generates the final value (to 2C standards) after getting an unsigned product back from the unsigned multiplier. This value is then loaded into the oFIFO when the data is determined to be formatted in 2C.

The SM circuit is the second RM and similar in many ways to the 2C RM. It takes the absolute values of the inputs and sends them to the unsigned multiplier (in the static portion of the circuit). However, it then generates the final value (to SM standards) after getting an unsigned product back from the unsigned multiplier. This value is then loaded into the oFIFO when the data is determined to be formatted in SM.

B. Implementing the AXI Interfaces

After having generated the necessary HDL components and determining their implementation, it was necessary to create an interface that would allow these circuits to communicate with the PS. An AXI4-Full interface was used for the main multiplier IP as this allows the most flexibility. Although the multiplier circuit is not pipelined and cannot make full use of the advanced benefits of AXI4-Full, it still allows an easier transaction to take place between the PS and PL.

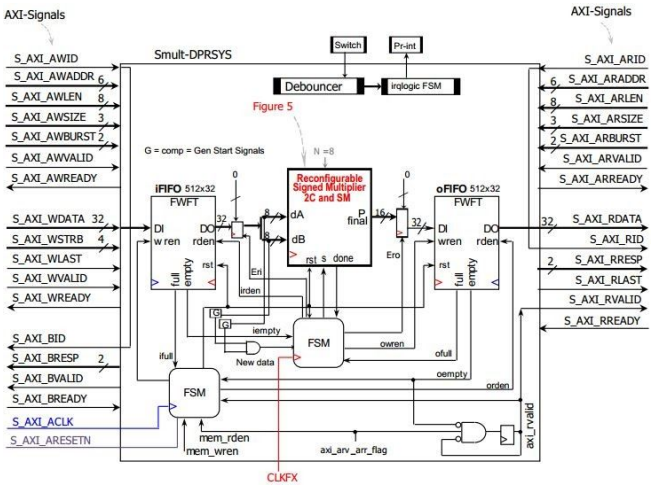

Fig. 6. Signed Multiplier AXI4-Full Implementation

Fig. 6 shows the full architecture of the signed multiplier block. It includes an input and output interface for AXI4-Full (along with an input and output FIFO), two FSM's to control communication and correct data processing, the signed multiplier logic, and also circuitry to generate a PL to PS interrupt. This interrupt is controlled by a state machine that monitors the state of a debounced button input and generates an interrupt when pushed. In order to be cleared, the software must request to read a value from a designated address from the peripheral. When this happens and the interrupt FSM realizes this, the interrupt that is going to the PS is then deasserted.

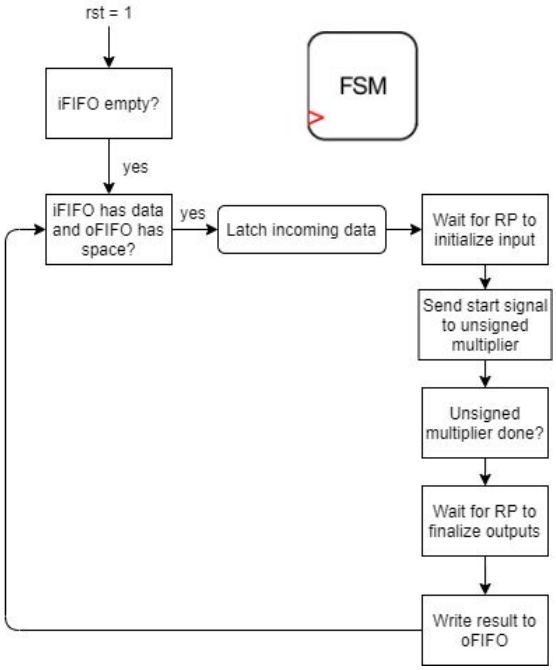

Fig. 7. High Level Application FSM

When the PS sends data to the PL to process (via the AXI4-Full bus), this data is stored into a FIFO (iFIFO). At this point, the peripheral has data that needs to be processed. After the system is initialized, this FSM (Fig. 7) begins to check if the iFIFO has data and if the output FIFO (oFIFO) has space to store more processed data. If both of these conditions are met, the incoming data is latched and the data sent to the core signed multiplier (as seen in Fig. 2). When the data has been completely processed by the internal core of the IP, the data is put into the oFIFO and then awaits the PS to request this data.

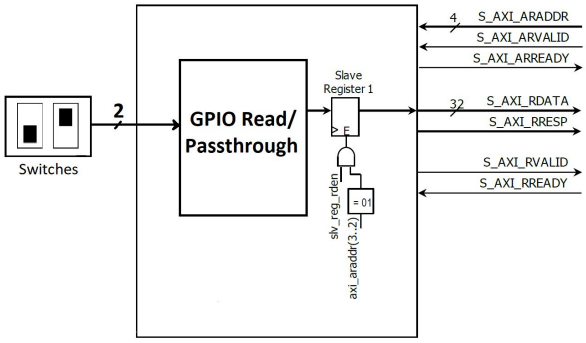

Fig. 8. GPIO Passthrough AXI4-Lite Implementation

In order to pass through values from switches to the PS, a custom AXI4-Lite GPIO peripheral was created. This circuit does not make use of any incoming data from the PS, as the only input data that is needed is from two switch inputs. These switch inputs are hardwired to the output register, which the PS has access to and can thus access at any time. As there is not much internal circuitry in this IP, there is no architecture diagram included.

C. *Software Functionality*

The software that is running on the PS is responsible for a few things in the full functional system. First, it is responsible for sending and receiving data from the signed multiplier IP. As the AXI4-Full implementation that is used makes use of 32-bit data buses, the 16 LSB's are used to transport the two 8-bit inputs to the multiplier. The software is also responsible for handling an interrupt (generated from a push button and modulated by the signed multiplier IP) from the PL which is an indication to perform a DPR. Before doing this reconfiguration, the value of a GPIO is checked to make a decision as to which circuit to load into the RP. It is at this point that the GPIO passthrough IP is interacted with and the value of a switch is polled to determine the circuit. A logic low corresponds to the 2C circuit and a logic high corresponds to the SM circuit.

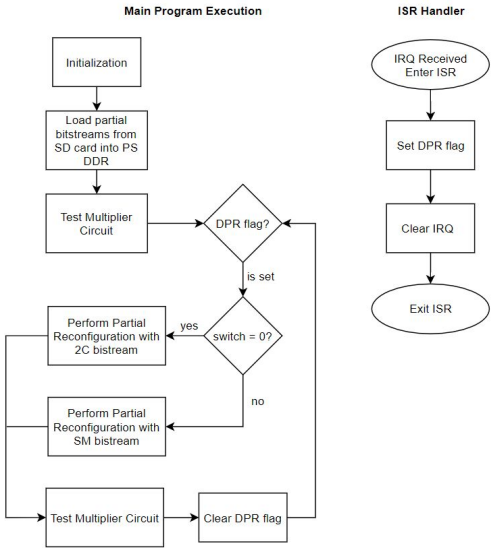

Fig. 9. Software Application Execution

Fig. 9 shows a very high level overview of the software that runs on the PS. It includes the initialization sequence (driver setup, loading partial bitstreams into memory, and a test of multiplier), the event loop, and the ISR handler.

Two other modes for the software can be achieved by editing a preprocessor statement in the source code that will change the software execution. One mode automatically performs two partial reconfigurations and automatically tests the multiplier circuit. The other mode only polls switch values to determine when to perform a DPR and which circuit to reconfigure to the RP.

III. Experimental Setup

Testbenches were created for all circuits and IP's to test their functionality. Vivado 2019.1 was used to run these simulations and the testbenches to verify the system behavior step by step. Any errors in the circuit components could be caught right away and removed before going through the process of generating the partial bitstreams.

In order to debug any issues in the software, it was possible to use the SDK terminal to communicate with the PS and determine what the software was executing. This was a later step and was only done once the hardware had been fully simulated and was verified to work after extensive testbench simulation.

IV. Results

The results obtained show that the system functioned as intended. All components and functionalities including partial reconfiguration, signed multiplication processing, PL to PS interrupt and interrupt handling (by software), along with switch reading work as intended. This allows for everything to come together and function. The two other software modes mentioned previously function as intended as well, although not being used.

In order to verify results (both during implementation and testing), it was necessary to generate input values and their corresponding output values for each circuit. These were used both throughout the simulation and verification process of hardware as well as during system testing when testing software and DPR functionality.

Table 1. Input and Output Values of Corresponding RM's

INPUT	OUTPUT	
	RP mode = "2C"	RP mode = "SM"
0x00000406	0x00000018	0x00000018
0x0000FD03	0x0000FFF7	0x00008177
0x0000FDFD	0x00000009	0x00003D09

Table 1 shows the sample inputs that were used for both testbench testing of the individual circuits and the values that were used by software when testing the system on the

Zybo Z7 board. The outputs correspond to the product of the input, but are in their respective number formats.

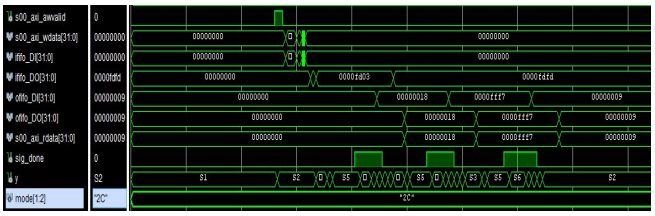

Fig. 10. 2C AXI Burst Simulation Results

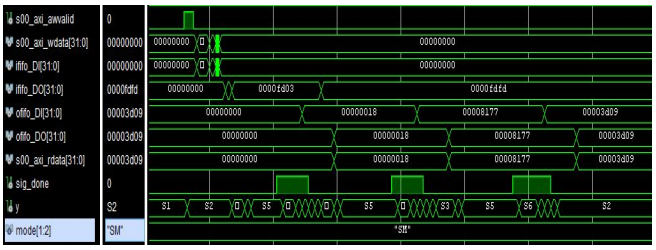

Fig. 11. SM AXI Burst Simulation Results

Fig. 10 and 11 both show a testbench simulation of an AXI master sending a burst of the three inputs as seen in Table 1 (although the last two inputs cannot be seen in the simulation window at this zoom). Following processing in each of the two circuits, the outputs are then loaded into the oFIFO and read again by the master. The values loaded into the oFIFO after being processed were compared with the output values in Table 1 and verified.

```
SD + SMULT Test:
*************************
(load_sd_to_memory): Loading 'mult_sm.bin' to memory
(load_sd_to_memory) : File Size: 117448 bytes
Close File: Success!

(load_sd_to_memory): Loading 'mult_2c.bin' to memory
(load_sd_to_memory) : File Size: 117448 bytes
Close File: Success!

smult_test
(smult_test) Signed Mult AXI4-Full Peripheral Test
Product = 0018
Product = FFF7
Product = 0009

start event loop


(ISR) PL Interrupt occurred!
isr done
(load_bit_to_pcap) IntrStsReg: F803000F:
(load_bit_to_pcap) Interrupt Status bits cleared! IntrStsReg: F803000F
(load_bit_to_pcap) DPR: Transfer to start: Source Address: 0011CC78...
(load_bit_to_pcap) DPR: Transfer completed!
--------------------
SMULT_SM CONFIGURED
--------------------
Asserting PR_reset.
(smult_test) Signed Mult AXI4-Full Peripheral Test
Product = 0018
Product = 8177
Product = 3D09


(ISR) PL Interrupt occurred!
isr done
(load_bit_to_pcap) IntrStsReg: F803300F:
PartialCfg = 1 (i.e., not 1st configuration)!
(load_bit_to_pcap) Interrupt Status bits cleared! IntrStsReg: F803000F
(load_bit_to_pcap) DPR: Transfer to start: Source Address: 0028AFE0...
(load_bit_to_pcap) DPR: Transfer completed!
--------------------
SMULT_2C CONFIGURED
--------------------
Asserting PR_reset.
(smult_test) Signed Mult AXI4-Full Peripheral Test
Product = 0018
Product = FFF7
Product = 0009
```

Fig. 12. Actual Demo Results

Fig. 12 shows a working version of the full system. First, the two partial bitstreams (mult_sm.bin and mult_2c.bin) are loaded into memory on the PS. A test of the multiplier is done and the event loop is then reached. Following this, the system waits for an interrupt to which it will then check the value of a switch through the GPIO IP and choose which partial bitstream to configure, following a test of the circuit after DPR. Fig. 12 shows this interrupt occurring twice, both times with the switch having different values (thus causing the two different bitstreams to be loaded each time).

Conclusions

Reconfigurable computing is a very good approach that can be used for many applications to reduce the cost by reducing hardware needs and providing a reconfigurable and flexible system. In this project, it was possible to build a reconfigurable system that performs signed multiplication in two ways (2C and SM) for two 8-bit inputs. Partial Reconfiguration along with the abilities of the Zynq-7000

chip and AXI allow for a very advanced system to be created. There are many applications beyond this that could host very advanced circuits (and thus very advanced RM's). Although the reconfigurable part of this circuit is rather simple, the most important part is showcasing the DPR functionality and the extensive capabilities that it allows. Aside from this, the use of a PL to PS interrupt is showcased as this gives even more flexibility in systems when needed.

As all necessary and wanted functionalities were achieved for this project, it is hard to come to a conclusion on any improvements to its functionality. One improvement could be changing up the interrupt clearing mechanism in the hardware. Although it functions, it is rather rudimentary at this point and a much more clean, efficient way could be developed to do this. Another improvement that could be made, if another system with more RM's was needed could be to parameterize the GPIO passthrough IP to allow more than just two I/O's to be polled by the software. Anywhere up to 32 I/O's (due to the 32-bit data bus used for this AXI implementation) could be used if this IP were changed to allow this.

This project posed many obstacles due to the current state of the world and the COVID-19 pandemic. Through a good system of communication and working on all items as needed, it was possible to finish without complications in the allotted amount of time. Advanced knowledge in DPR and the AXI protocol were developed through designing this system. Other knowledge attained includes the ability to work together remotely on a team, along with designing and validating a system through rigorous testing. Through all of the skills that were required to develop and consistent work, it was possible to finish the system and meet all requirements.

References

[1] https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug909-vivado-partial-reconfiguration.pdf

[2] https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug947-vivado-partial-reconfiguration-tutorial.pdf

[3] D. Llamocca, "Reconfigurable Computing" Oakland University. 2020.