View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by NASA Technical Reports Server

Optimization of the Multi-Spectral Euclidean Distance Calculation for FPGA-based Spaceborne Systems

Alejandro Cristo[1], Kevin Fisher[3], Rosa M. Pérez[1], Pablo Martínez[1], Anthony J. Gualtieri[2,3]

[1] *Departamento de Tecnología de los Computadores y de las Comunicaciones. Escuela Politécnica de la Universidad de Extremadura, Av/ Universidad s/n, 10005, Cáceres, Spain*
[2] *LuxAnalytica, Washington, DC, 20002, USA*
[3] *Software Engineering Division, NASA Goddard Space Flight Center, Greenbelt, MD, 20771, USA*

Corresponding author: acristo@unex.es

Abstract.

Due to the high quantity of operations that spaceborne processing systems must carry out in space, new methodologies and techniques are being presented as good alternatives in order to free the main processor from work and improve the overall performance. These include the development of ancillary dedicated hardware circuits that carry out the more redundant and computationally-expensive operations in a faster way, leaving the main processor free to carry out other tasks while waiting for the result. One of these devices is SpaceCube, a FPGA-based system designed by NASA. The opportunity to use FPGA reconfigurable architectures in space allows not only the optimization of the mission operations with hardware-level solutions, but also the ability to create new and improved versions of the circuits, including error corrections, once the satellite is already in orbit. In this work, we propose the optimization of a common operation in remote sensing: the Multi-Spectral Euclidean Distance calculation. For that, two different hardware architectures have been designed and implemented in a Xilinx Virtex-5 FPGA, the same model of FPGAs used by SpaceCube. Previous results have shown that the communications between the embedded processor and the circuit create a bottleneck that affects the overall performance in a negative way. In order to avoid this, advanced methods including memory sharing, Native Port Interface (NPI) connections and Data Burst Transfers have been used.

1. Introduction.

Modern satellites carry and will carry scientific instruments, including hyperspectral cameras, that generate data at unprecedented rates, far exceeding the bandwidth of radio links to the ground. This makes it impractical to rely on ground-based computers to process and extract information from the data. Often the value of the satellite is not in the sheer volume of data it collects, but in the small but critical bits of information extracted from that data: early warning of an erupting volcano or forest fire, or detection of flooding or deforestation through image segmentation. If the processing needed to produce that information can be done on the spacecraft, it can be sent much more quickly and economically to users on Earth.

In this way, a very important part in the design of future space missions is the way data is computed and transferred, trying to maximize the spaceborne computation capabilities and minimize the quantity of data generated and sent to Earth. Several missions currently developed by different space agencies are trying to leverage this concept. One example is the NASA HyspIRI mission [1-2], which expects about 3.5 – 4.6 Tb of generated data per day to study Earth events including wildfires, volcanoes and drought. Another instance is the future ESA Sentinels [3], a total of five individual missions involving six satellites so far (Sentinel-1, with its launch planned in 2013, will be the first one to be in orbit). These satellites will provide scientific information for land and ocean

studies and analysis, generating and transferring more than 200 Gb of data every day.

In this context, the architecture used to pre-process and prepare the data before being transmitted to Earth is very important. The challenge is not limited to merely sending the information acquired by the on-board instruments, but in generating reliable spaceborne results to minimize the total data to transfer and the delay in producing real-time products, and to develop alert and monitoring systems. These artifacts require new and advanced hardware architectures with high-frequency processor systems and high-capacity (and yet fast) memories. These spaceborne ancillary devices are very useful to help the main computer complete its tasks, generate near-production-quality results, and prepare them in an organized way to be transferred to Earth.

Satellite on-board processing has many challenges. Flying on a spacecraft, computers are constrained in size, mass, and maximum power consumption. They must also operate in the presence of radiation strong enough to cause logic and data storage errors, even in Low Earth Orbit (LEO) [4-5]. This often requires redundancy or additional hardware that further constrains the computer's capabilities [6].

One of the more relevant proposals in the spaceborne processing field is the SpaceCube (Figure 1) system [7], a high performance re-configurable system family designed and implemented at NASA Goddard Space Flight Center. Its design allows it to work in space-based applications that require on-board processing with very low energy consumption.

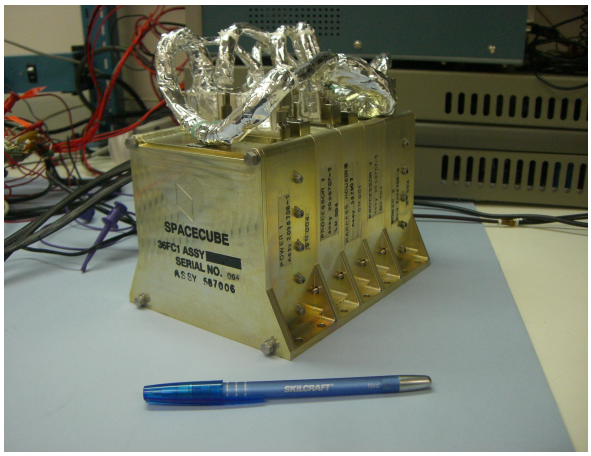

Figure 1: SpaceCube v1.0b Flight System.

Spacecube is a solution to many of these challenges: a small, efficient, radiation-tolerant, yet powerful on-board computer. It combines a low-power, general-purpose central processing unit (CPU) with a field-programmable gate array (FPGA). If allocated wisely, the FPGA fabric can host hardware circuits that implement data processing algorithms as fast, or perhaps faster, than desktop or server systems on the ground.

The first version of SpaceCube (SpaceCube 1.0a) was based on the Xilinx Virtex-4 commercial FPGA [8], and was first flown in space aboard STS-125, the Hubble Space Telescope Servicing Mission 4 (HST-SM4). A second unit (SpaceCube 1.0b) was installed on the International Space Station (ISS) in November 2009 as part of the Materials International Space Station Experiment 7 (MISSE7) [9] experiment payload, and is currently serving as an on-orbit SpaceCube technology validation testbed.

The current, second-generation version of SpaceCube (SpaceCube 2.0) is based on the Xilinx Virtex-5 FPGA [10] and can be populated with either a commercial device paired with the upset mitigation strategies being tested on MISSE7, or with the radiation-hardened Virtex-5 QV FPGA

part (which does not include the dual PowerPCs).

Hybrid CPU-FPGA algorithms are a unique design problem [11]. Work must be divided intelligently between the CPU and FPGA, with either a communication bus or shared memory block for message and data passing. Both devices are capable of pipelining and parallel processing, but to different extents. Efficient algorithms often employ all of the above, weaving them together without the benefit of an operating system or high-level services, which can consume and re-direct resources.

This paper describes a very common operation for remote sensing data processing called Multispectral Euclidean Distance Calculation (MEDiC), a hybrid CPU-FPGA algorithm developed to run on Virtex-5 FPGAs, and thus on SpaceCube 2.0.

Section 2 describes in detail two different versions of the algorithm, each using different connections and configurations to a shared memory. Following this, Section 3 shows the results of experiments to test our designs, including the speed-up, maximum frequencies and communication time analysis. Section 4 analyses the results and overall speedup, and makes concluding remarks.

2. Methodologies.

In order to optimize the global performance of hyperspectral image processing methodologies, we have designed a circuit capable of computing the Euclidean distance between two spectra. This time-consuming operation is widely used by many image processing techniques, so the exercise of adapting it to hardware would enable a decrease in the execution time of many common algorithms. One example of this is the Hierarchical SEGmentation (HSEG) algorithm [12-13], which segments hyperspectral images, and spends over 90% of its overall computation time on calculating multi-spectral Euclidean distances. Designing a hardware module for this operation in a Xilinx Virtex-5 FPGA would lead to its direct integration in a SpaceCube system, so it could be performed on-board a satellite to generate real-time results.

The Euclidean distance operation between two spectra A and B has been designed as a hardware module called Multispectral Euclidean Distance Calculator (MEDiC), which is attached to a processor-based system as a peripheral through a Processor Local Bus (PLBv46) [14], and where other peripherals (including a PowerPC processor running up to 400 MHz) and a bus arbiter are present (Figure 2). Furthermore, the designed peripheral, along with the MEDiC module, has registers for peripheral-processor communications, and a Control Logic that ensures that messages are sent and received correctly.

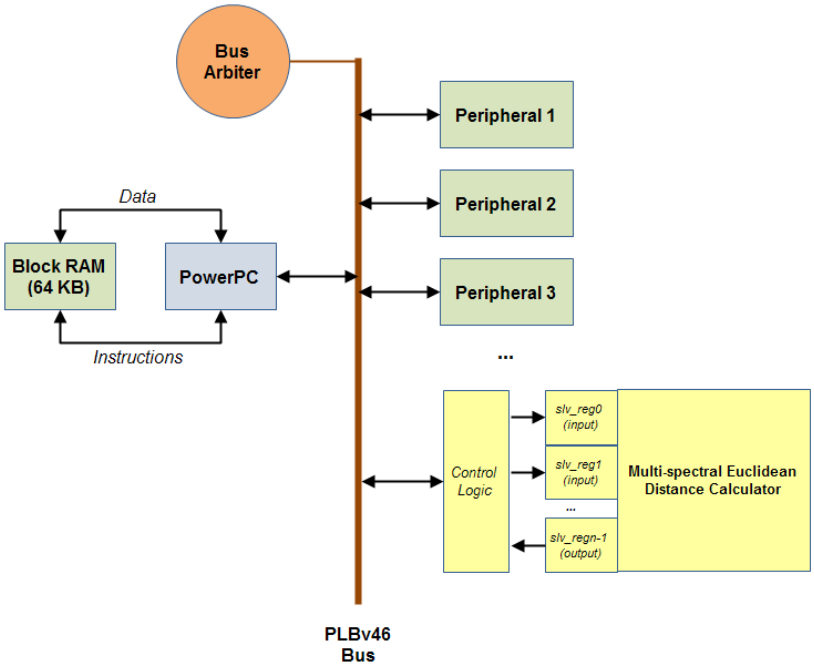

Figure 2. Diagram showing the Processor-Peripheral connection through the PLBv46 bus.

One of the options to transfer the spectra from the processor to the peripheral and the results from the peripheral to the processor is through the PLBv46 bus. This may create bottlenecks, however, since it is also used by the other peripherals. In order to avoid this, a 64-bit, 128MB DDR2 memory will be used to store both spectra and result. In doing so, the PowerPC processor and the MEDiC peripheral will be directly connected to the memory through the Multi-Port Memory Controller (MPMC) component [15]. This component offers a total of eight independent ports to transfer data in parallel using a Native Port Interface (NPI) connection, the highest speed connection to the MPMC component offered by the Xilinx board.

Two different designs are considered, which differ in the way the PowerPC processor and peripheral are attached to the DDR2 memory through the MPMC component:

- *Design 1*: One port is dedicated to the PowerPC processor, and the seven remaining will be dedicated to the MEDiC peripheral. In this last case, the read or write of one word will take several clock cycles.

- *Design 2*: One port is dedicated to the PowerPC processor, and four are dedicated to the MEDiC peripheral. In this last case, the write of a word will take several clock cycles, but the *Read Burst Mode* will be activated to allow reads of several-word blocks in only one clock cycle.

With this design, all spectrum transfers and results are performed through the MPMC ports, while the PLBv46 is used to transfer messages from the processor to signal the MEDiC peripheral to start working, and from the MEDiC peripheral to alert the processor that the final result is ready to read from memory.

2.1. *Design 1* of the hardware architecture.

In this first design, one port of the DDR2 memory is used by the PowerPC processor, and the MEDiC peripheral makes use of the seven remaining ports (Figure 3). Furthermore, it is necessary

to take into account the following considerations:

- The NPI connection has a width of 64 bits. Since the spectra are represented by 32-bit values, every time a read is carried out, two 32-bit spectrum values are received at the same time. This decreases the read time.

- The MEDiC will use the seven ports remaining to read the values from memory in parallel. There are 64 values to read in total (32 for spectrum *A* and 32 for spectrum *B*), so the first four ports will perform five reads apiece (obtaining 10 spectrum values each, or 40 total, since it works with 64-bit port connections), whereas the last three ports will each perform four reads (obtaining eight values each, or 24 total, due to the same issue). Moreover, the last port (Port7) must be capable of writing the final result in memory.

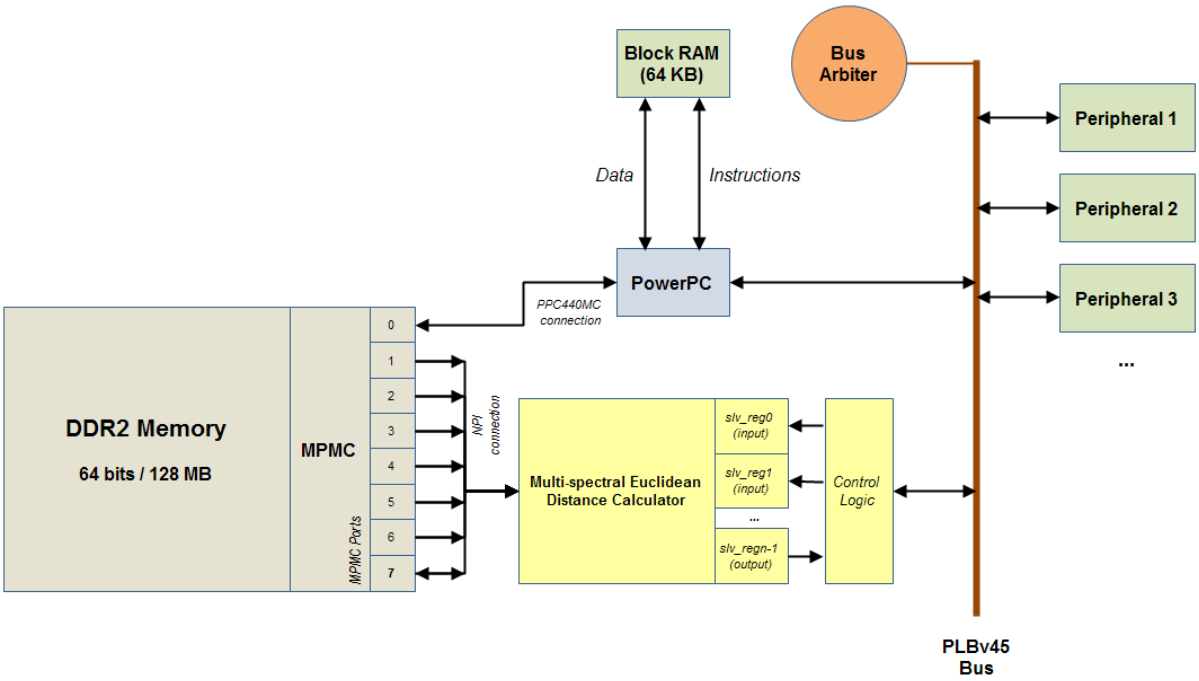

Figure 3: DDR2 Memory-Peripheral-Processor connection diagram for the Design 1, where the MEDiC peripheral makes use of the seven remaining ports by using a 64-bit NPI connection.

Additionally, the MEDiC peripheral is composed of a set of registers to reset the entire circuit (*Reset*), to allow the processor to start the MEDiC component (*Enable*), and to allow the MEDiC peripheral to indicate to the processor the execution has finished and that it can find the final result in a certain memory address (*Done*) (Figure 4).

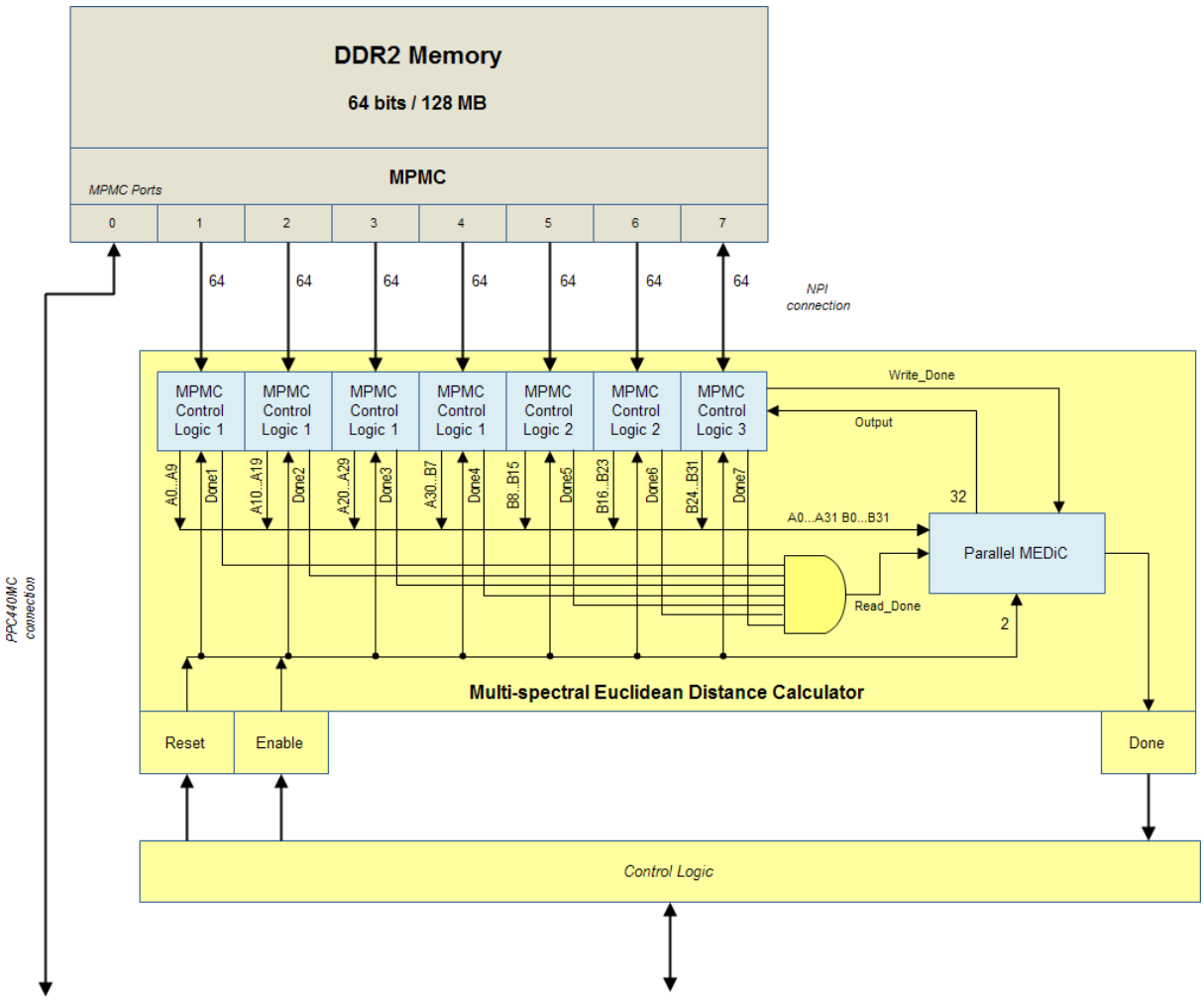

Figure 4: Diagram of the MEDiC peripheral, showing the logic modules to control the MPMC component, the connections to the seven remaining MPMC ports, and the conventional communications by using registers.

The MEDiC peripheral is mainly composed of the following modules and registers:

- MPMC Control Logic: Module that transfers data from a certain memory port. There are three different MPMC Control Logic modules, according to the number of read to be performed in each port, and whether it is necessary to read and write, or only read:
  - MPMC Control Logic 1: performs five 64-bit reads from memory (obtaining ten 32-bit spectrum values). This logic will be used by ports from 1 to 4.
  - MPMC Control Logic 2: performs four 64-bit reads from memory (obtaining eight 32-bit spectrum values). This logic will be used by ports 5 and 6.
  - MPMC Control Logic 3: performs four 64-bit reads from memory (obtaining eight 32-bit spectrum values). Furthermore, it writes the resulting Euclidean distance in memory. This logic will be used by the last port (Port7).
- Parallel MEDiC: Module that performs the multi-spectral Euclidean distance calculation in parallel.
- Enable Register: The processor stores a 1 in this register to command the MEDiC peripheral

to start working.

- Reset Register: The processor will store a 1 in this register to command the MEDiC peripheral to reset all its modules and signals.

- Done Register: The MEDiC peripheral will store a 1 in this register when its execution has finished and the result of the Euclidean distance calculation is available in memory.

- Control Logic: This module handles all the MEDiC peripheral – processor communications, and stores all the convenient values in their corresponding communication registers.

2.1.1. The MPMC Control Logic 1.

The MPMC Control Logic 1 module (Figure 5) reads ten spectrum values from a certain memory port. Since the NPI connection works with 64 bits, and each read will obtain two spectrum values at the same time, it will be necessary to perform a total of five reads. It is composed of the following modules:

- A 3-bit hardware counter that increases by one every time a read has been performed. Since it works with three bits, it can count up to 8.

- The central logic composed by a NAND, an OR, a NOT and an AND gate that controls the reads from memory. It is enabled when the counter is less than 5, the MPMC Control Logic 1 is enabled for first time (signal *E* active for only one clock cycle), and the MPMC Controller has finished reading (it should not execute while this component is working). Once the counter reaches 5 (input 100 for the NAND gate), this central logic avoids the MPMC Control Logic 1 to continue working and the execution finishes.

- A 3-bit decoder (DEC). Only one of its output signals is active according to the counter. Since only five reads will be performed, the three last output signals are not used.

- Program Counter (PC): A simple 32-bit counter that increases the base read address (*Address_Read*) by 8 bytes (64 bits). In each execution (read performed), the address points to the position in memory of the new value to read. Each read address is sent to the MPMC Controller module to carry out the read of the corresponding value.

- The registers R0...R9 store the ten values read from memory. It is necessary to consider that the data read from memory returns 64 bits, so it contains two 32-bit spectrum values in the same packet. In this way, the read data is split into two values; the first one is stored in the corresponding right register, and the second one in the corresponding left register. The AND gate enables both right and left registers. The RDone register becomes active (stores a 1) when the last two values (the last read) have been stored in R8 and R9. This generates an output of the MPMC Control Logic module that signals that the reads of the ten spectrum values have finished.

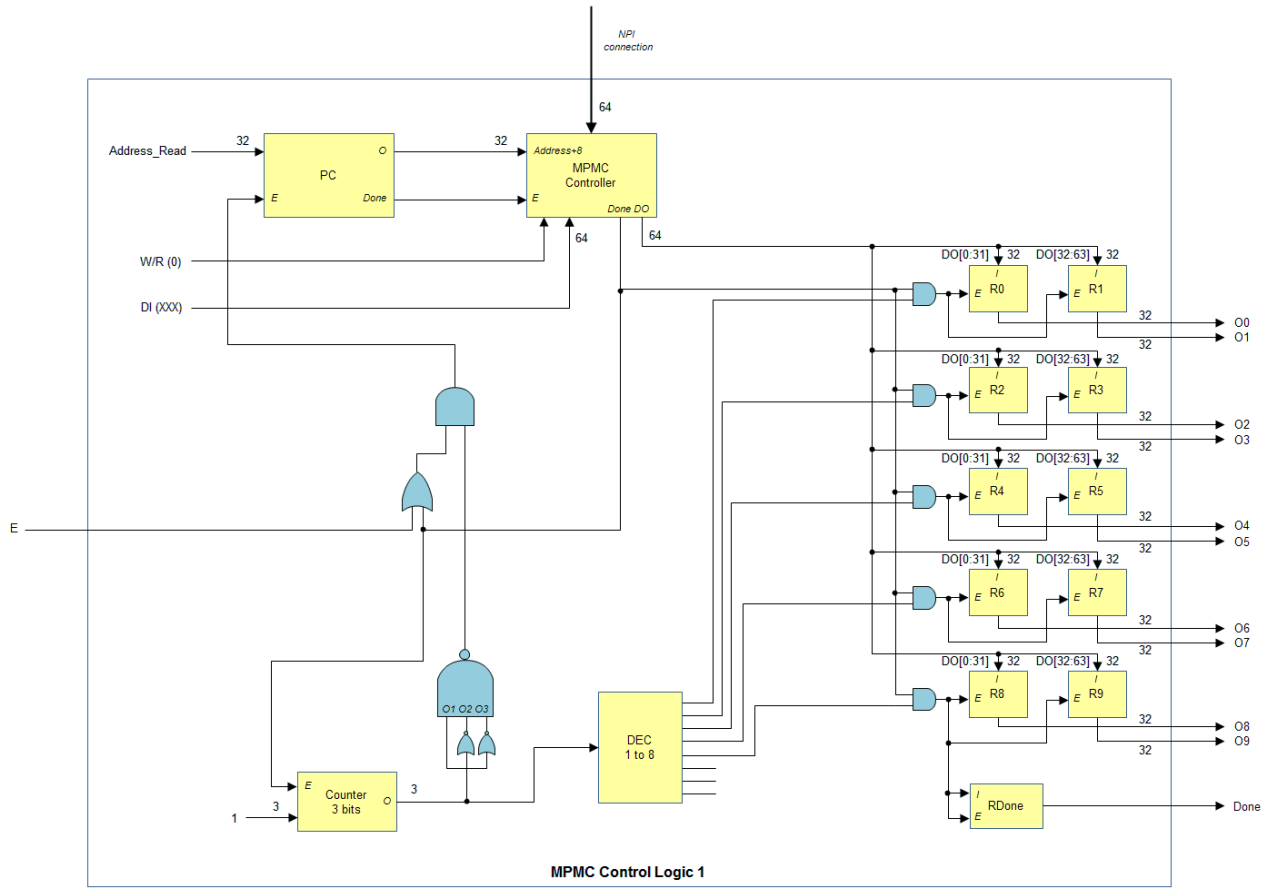

Figure 5: Diagram of the MPMC Control Logic 1 module, responsible for obtaining the ten values from memory through each port.

2.1.1.1. The MPMC Controller.

This module handles the MPMC component, and has been implemented as a state machine that reads and writes data. The MPMC component is an interface provided by Xilinx to use the DDR2 memory. These signals are shown in Figure 6a and Figure 6b for writing and reading actions, respectively. The MPMC Controller handles these signals when reading or writing a value from memory is required.

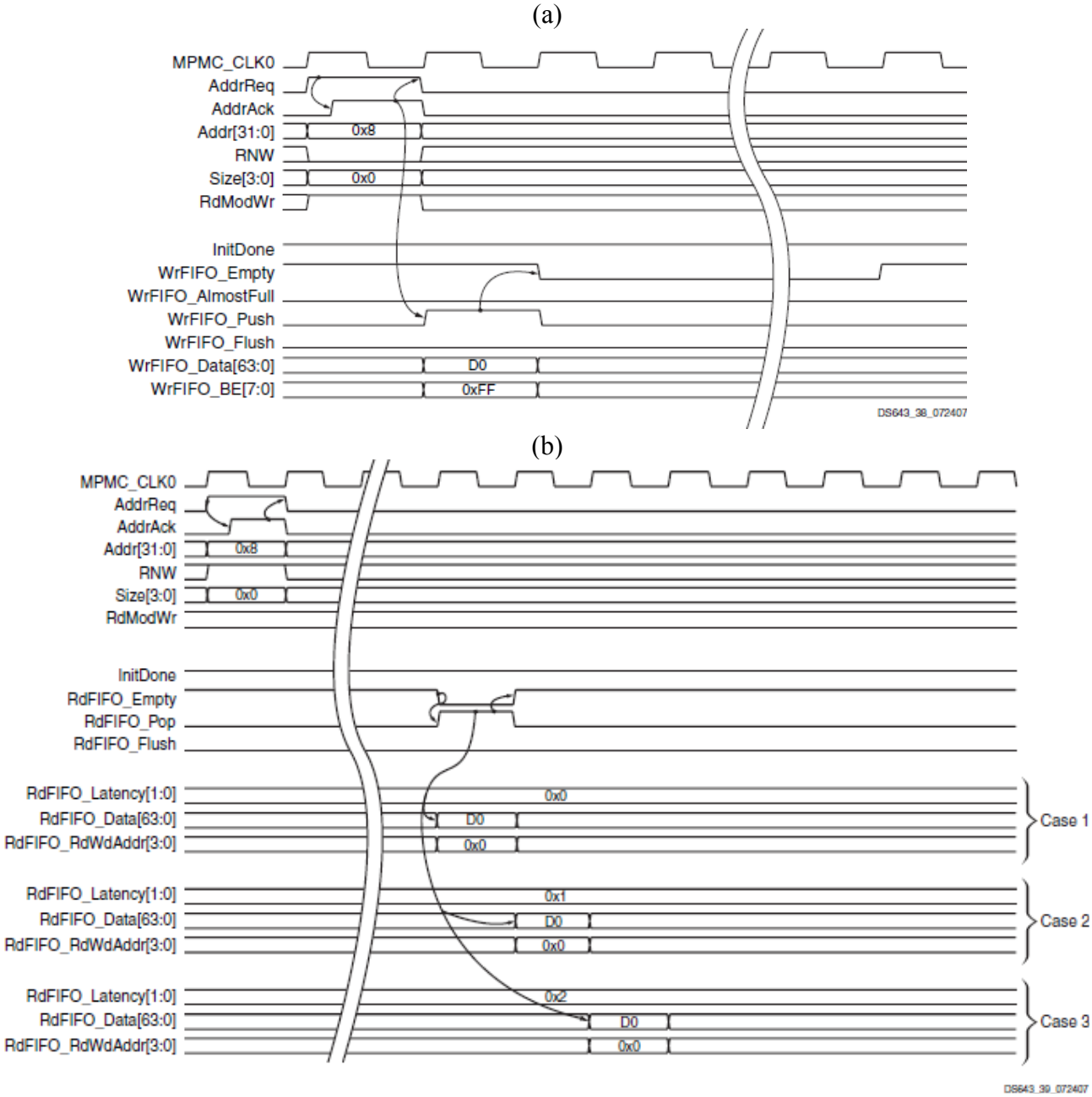

Figure 6: MPMC module control signals. (a) Signals of the MPMC module to handle a write of a 64-bit value in memory. (b) Signals of the MPMC module to handle a read of a 64-bit value from memory.

Once the action required (read or write) has been carried out, the component outputs the *Read_Done* or *Write_Done* signal.

2.1.2. The MPMC Control Logic 2.

The MPMC Control Logic 2 module (Figure 7) reads 8 spectrum values from a given memory port. Since the Native Port Interface (NPI) connection works with 64 bits, and each read will obtain two spectrum values at the same time, it will necessary to perform a total of 4 reads. This module has the same design as the MPMC Control Logic 1 module (Figure 4), but with the following modifications:

- A 2-bit hardware counter that increments every time a read has been performed. Since it

works with two bits, it can count up to 4.

- A central logic composed of a NAND, an OR and an AND gate that ensures that reading from memory is enabled when the counter is less than 4, the MPMC Control Logic 2 is enabled for first time (signal *E* is active only for one clock cycle), and the MPMC Controller has finished reading (it should not execute while this component is working). Once the counter reaches 4 (input 11 for the NAND gate), this central logic stops the MPMC Control Logic 2 and the execution finishes.

- A 2-bit decoder (DEC), which tracks the number of reads performed, and enables only one set of registers at a time.

- Registers R0...R7 to store the 8 values read from memory.

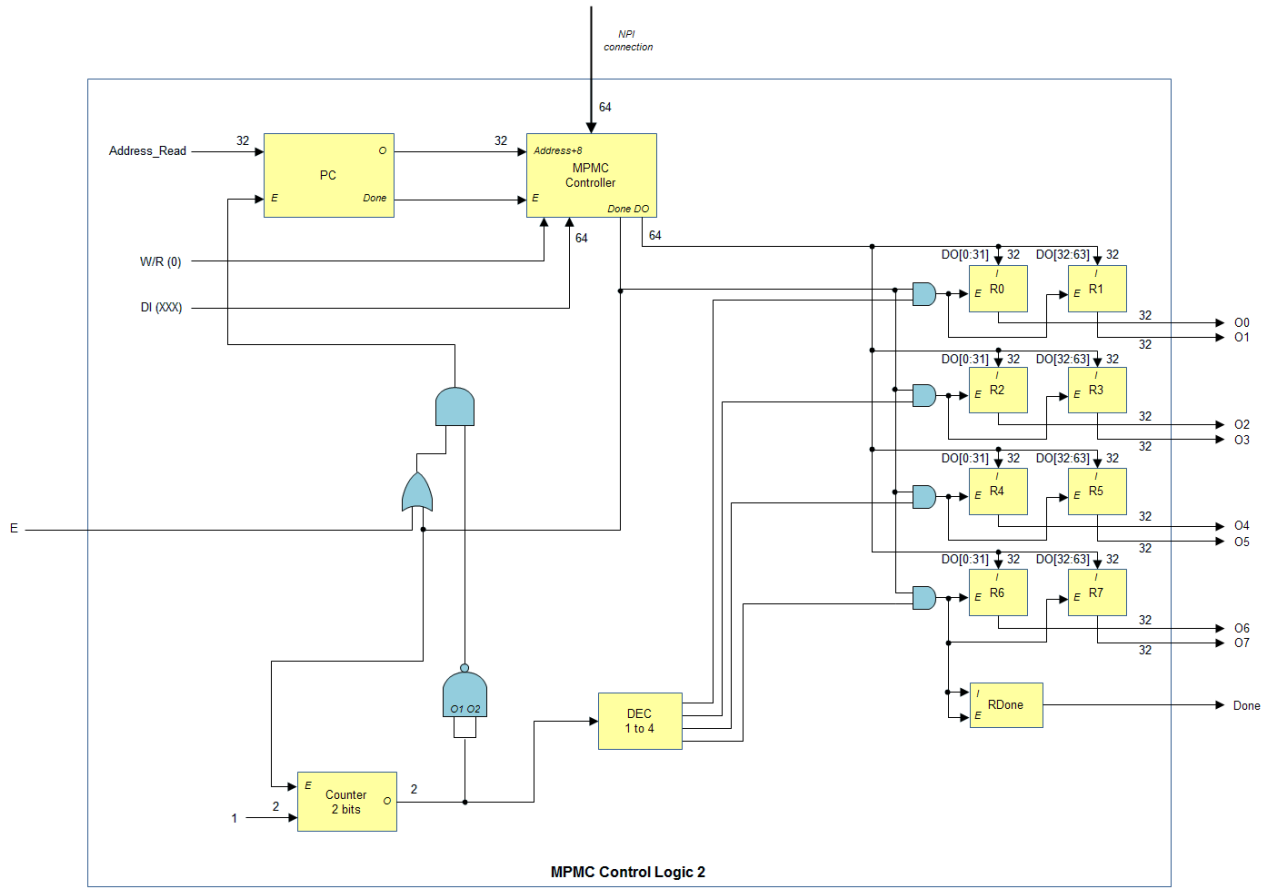

Figure 7: Diagram of the MPMC Control Logic 2 module, responsible of obtaining the eight values from memory for each port.

2.1.3. The MPMC Control Logic 3.

The MPMC Control Logic 3 module (Figure 8) reads eight spectrum values from memory, and writes the result of the Euclidean distance calculation into memory. Since the NPI connection is 64 bits wide, and each read will obtain two spectrum values at the same time, it will perform a total of four reads. The result of the calculation will be stored in a 64-bit format. This module has the same design as the MPMC Control Logic 2 module (Figure 7), but with the following modifications:

- Two 32-bit multiplexers (MUX) choose between the program memory address (from PC), in the case of the reads (W/R=1), or a defined memory address to write the result (W/R=0).

- A *WDone* register that drives an output of the MPMC Control Logic module, indicating that the reads of the 8 spectrum values have finished.

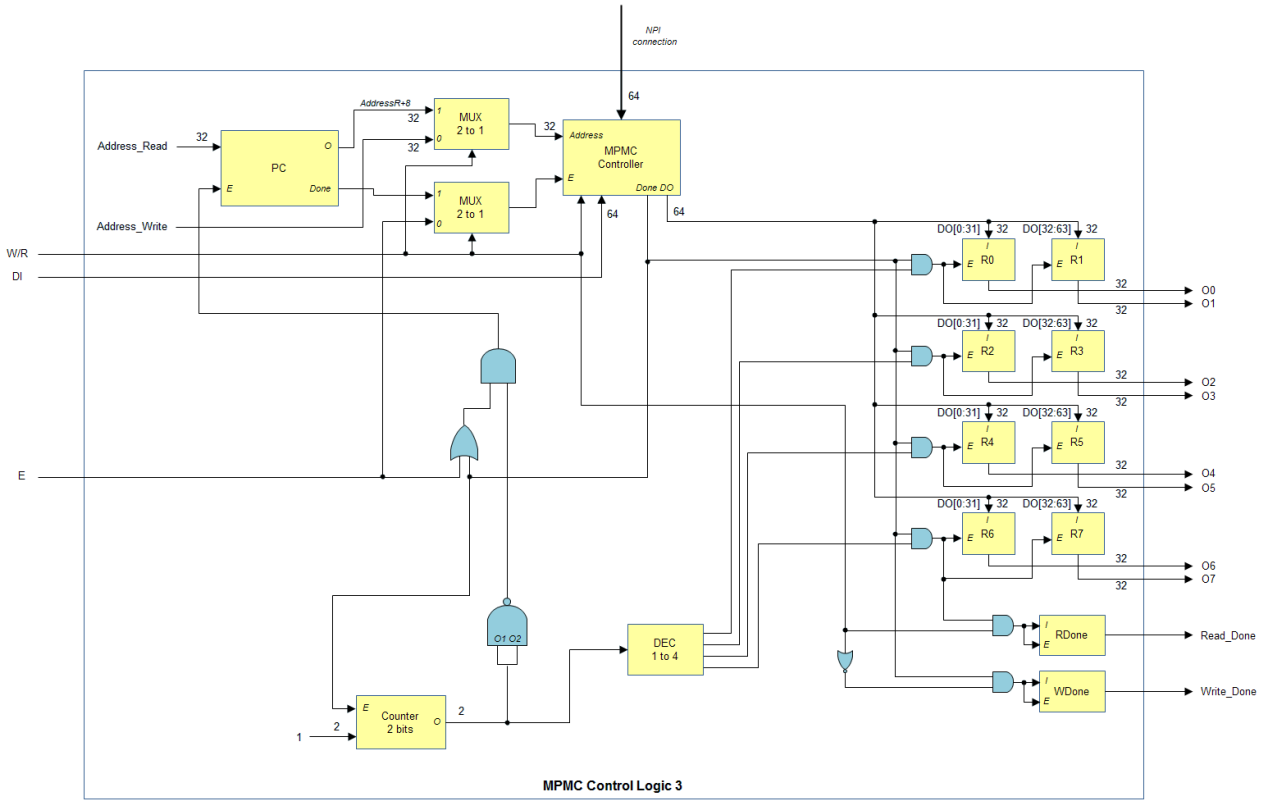

Figure 8. Diagram of the MPMC Control Logic 3 module, responsible of obtaining the 8 values from memory for each port, and writing the result of the Euclidean distance calculation.

2.1.4. The Parallel MEDiC.

The Parallel MEDiC module performs the Euclidean distance calculation between two spectra A and B, according to Equation 1, where A_i and B_i are the values of the spectra along the bands.

$$d(A,B)=\sqrt{(A_1-B_1)^2+(A_2-B_2)^2+\ldots+(A_n-B_n)^2} \quad (1)$$

Since the spectra have 32 bands, this module, shown in Figure 9, uses the following components:

- 32 subtraction units to carry out the A_i-B_i operations in parallel.

- 32 multipliers to carry out the $(A_i\text{-}B_i)^2$ operations in parallel.

- Five levels of adders to perform the $(A_1\text{-}B_1)^2+(A_1\text{-}B_1)^2+\ldots+(A_n\text{-}B_n)^2$ sum quickly, in parallel.

- An accumulator that stores the $(A_1\text{-}B_1)^2+(A_1\text{-}B_1)^2+\ldots+(A_n\text{-}B_n)^2$ sums, to support a case where more than 32 bands are required and multiple iterations of the circuit are needed.

- Eight latches that will delay the *Read_Done* signal for 8 clock cycles. *Read_Done* indicates that all the spectra have been read fro mmemory, and they are ready to be used for the Euclidean distance calculation. After 8 more clock cycles, the result of the sum will be ready, and this delayed signal will prompt the accumulator to store the result (and avoid storing non-valid data while the sum is being performed).

- A SQRT module that will perform the square root of $(A_1\text{-}B_1)^2+(A_1\text{-}B_1)^2+...+(A_n\text{-}B_n)^2$ providing the final result after 10 clock cycles. It makes use of the CORDIC algorithm [16-17].

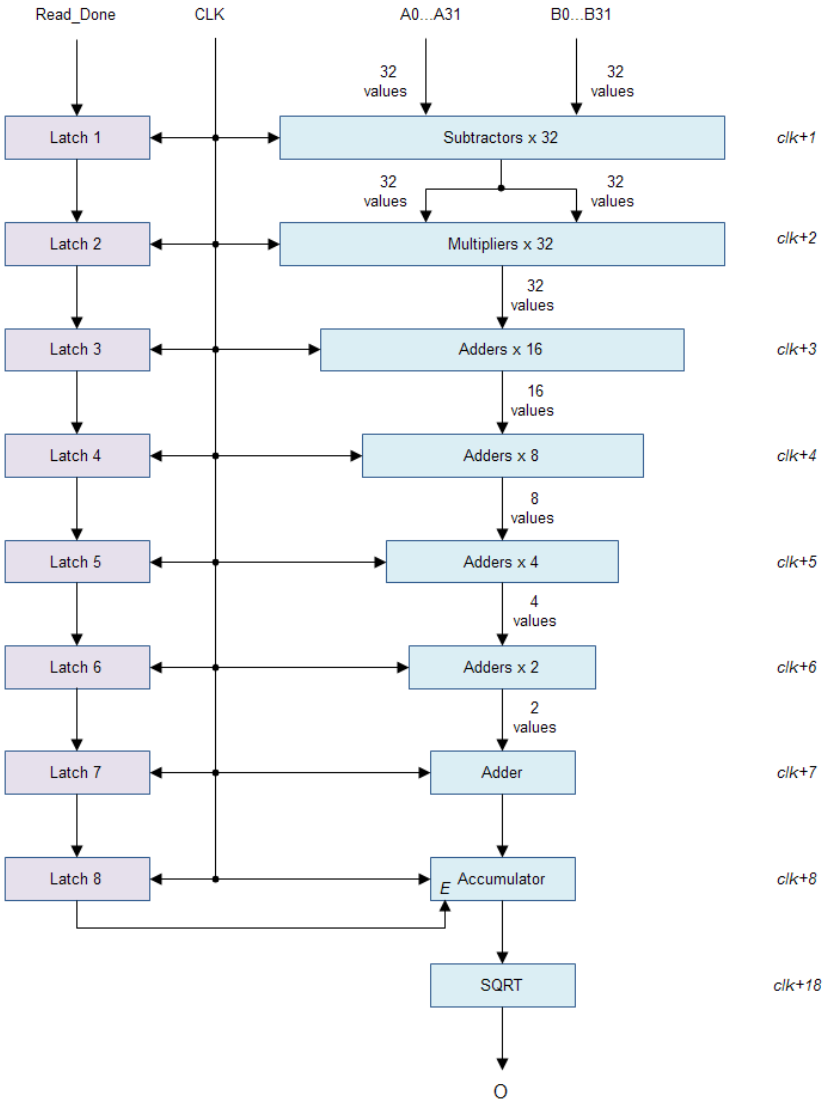

Figure 9: Diagram of the MEDiC module, responsible for calculating the Euclidean distance between two spectra A and B.

The total number of clock cycles required for the MEDiC module to perform the Euclidean distance calculation of two n-band spectra is defined by Equation 2, where $SUBS_{cc}$, $MULS_{cc}$, $ADDS_{cc}$, ACC_{cc} and $SQRT_{cc}$ are the time in clock cycles for subtraction, multiplication, addition, accumulation and square root, respectively.

$$Total_{cc}=(SUBS_{cc}+MULS_{cc}+ADDS_{cc}+ACC_{cc})+\left(\left\lceil\frac{n}{32}\right\rceil-1\right)+SQRT_{cc}=\left\lceil\frac{n}{32}\right\rceil+17 \quad (2)$$

2.2. *Design 2* of the hardware architecture.

The previous design used the DDR2 memory through the MPMC module, which offered a total of 8 ports to transfer the data to and from memory by using a NPI connection. Reading or writing a value in memory requires several tens of clock cycles, depending on how busy the MPMC module is. Xilinx provides a way to improve the transfer times (very useful to optimize read operations in the proposed system) by using the NPI in Burst Read mode. If the Burst Read mode is enabled, and one indicates the base read address, a full block of memory can be read at the rate of one word per clock cycle. The limitation of this mode is that the number of words in a block must be a power of two. The proposed design must read a total of 64 values from a 64-bit memory (obtaining two values per read), so there are different options according to the number of ports to use:

- If 1 MPMC port is used, it would be necessary to read one 32-word block (64 values). Total execution time: 32 cc.

- If 2 MPMC ports are used, each port would read one 16-word block (32 values). Total execution time: 16 cc.

- If 3 MPMC ports are used, two ports would read one 11-word block (22 values) and one port would read one 10-word block (20 values). However, since block size must be power of 2, the first two ports should read one 16-word block (32 values) whereas the last one would read nothing. Total execution time: 32 cc.

- If 4 MPMC ports are used, each port would read one 8-word block (16 values). Total execution time: 8 cc.

- If 5 MPMC ports are used, four ports would read one 7-word block (14 values) and one port would read one 6-word block (12 values). Since block size must be a power of 2, the first four ports would read one 8-word block (16 values) whereas the last one would read nothing. Total execution time: 8 cc.

- If 6 MPMC ports are used, five ports would read one 6-word block (12 values) and one port would read one 2-word block (4 values). Since block size must be a power of 2, the first four ports would read one 8-word block (16 values) whereas the last two would read nothing. Total execution time: 8 cc.

- If 7 MPMC ports are used, six ports would read one 5-word block (10 values) and one port would read one 2-word block (4 values). Since block size must be a power of 2, the first four ports would read one 8-word block (16 values), and the last three would read nothing. Total execution time: 8 cc.

So, the optimal option would be to use four MPMC ports to read the values from memory, and a fifth port to store the final result calculated by the Parallel MEDiC module (Figure 10). There is no need to enable the Burst Write mode for the fifth port, since only one word (the result of the Euclidean distance calculation) will be stored.

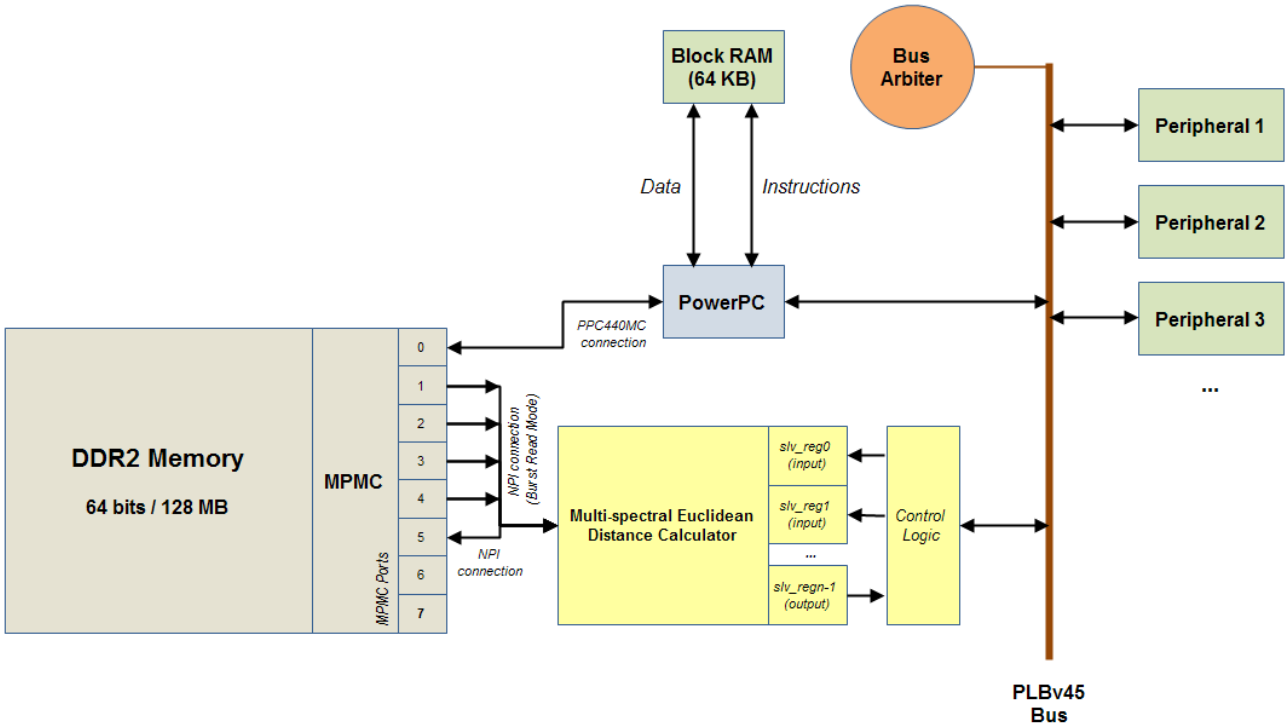

Figure 10: DDR2 Memory-Peripheral-Processor connection diagram for Design 2, where the MEDiC peripheral makes use of four MPMC ports with the Burst Read mode enabled for reading data, and a fifth one for writing the result. All the ports use 64-bit NPI connections.

The MEDiC peripheral is composed of the same modules as in the first design (Figure 4), except it only uses five ports, and the MPMC Control Logic has been modified (Figure 11).

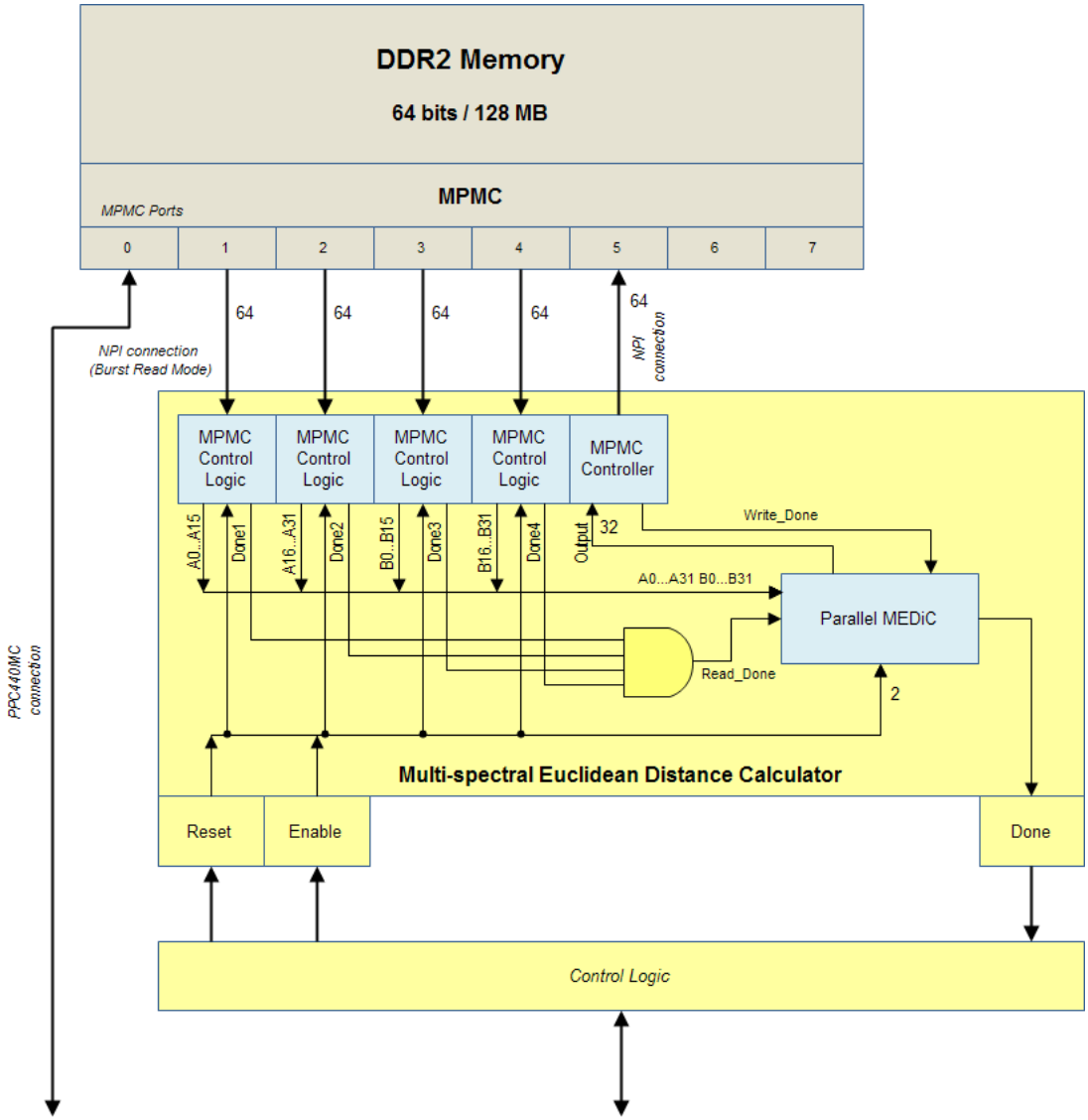

Figure 11: Diagram of the MEDiC peripheral, where the logic modules to control the MPMC component, the connections to the five MPMC ports, and the conventional communications by using registers are shown.

2.2.1. The MPMC Control Logic.

The MPMC Control Logic module (Figure 12) reads 16 spectrum values from a certain memory port using the Burst Read mode. Since the NPI connection is 64 bits wide, and each read will obtain two spectrum values at the same time, it will be necessary to perform a total of eight reads. This module has the same design as the MPMC Control Logic 1 module (Figure 4), but with the following modifications:

- A 3-bit hardware counter that increments (up to 8) every time a read is performed.

- A central logic composed by a NAND, an OR and an AND gate that controls the reads from memory. Reads are enabled when the counter is less than 8, the MPMC Control Logic is enabled for first time (signal *E* active only for one clock cycle), and the MPMC Controller has finished reading (it should not execute while this component is working). Once the counter reaches 7 (input 111 for the NAND gate), this central logic stops the MPMC Control Logic and the execution finishes.

- A 3-bit decoder (DEC). Only one of its output signals is active, determined by the counter, and so, each of the data values read is directed to a unique register.

- The number of registers has increased to 16 (R0... R15), in order to store the 16 values from memory.

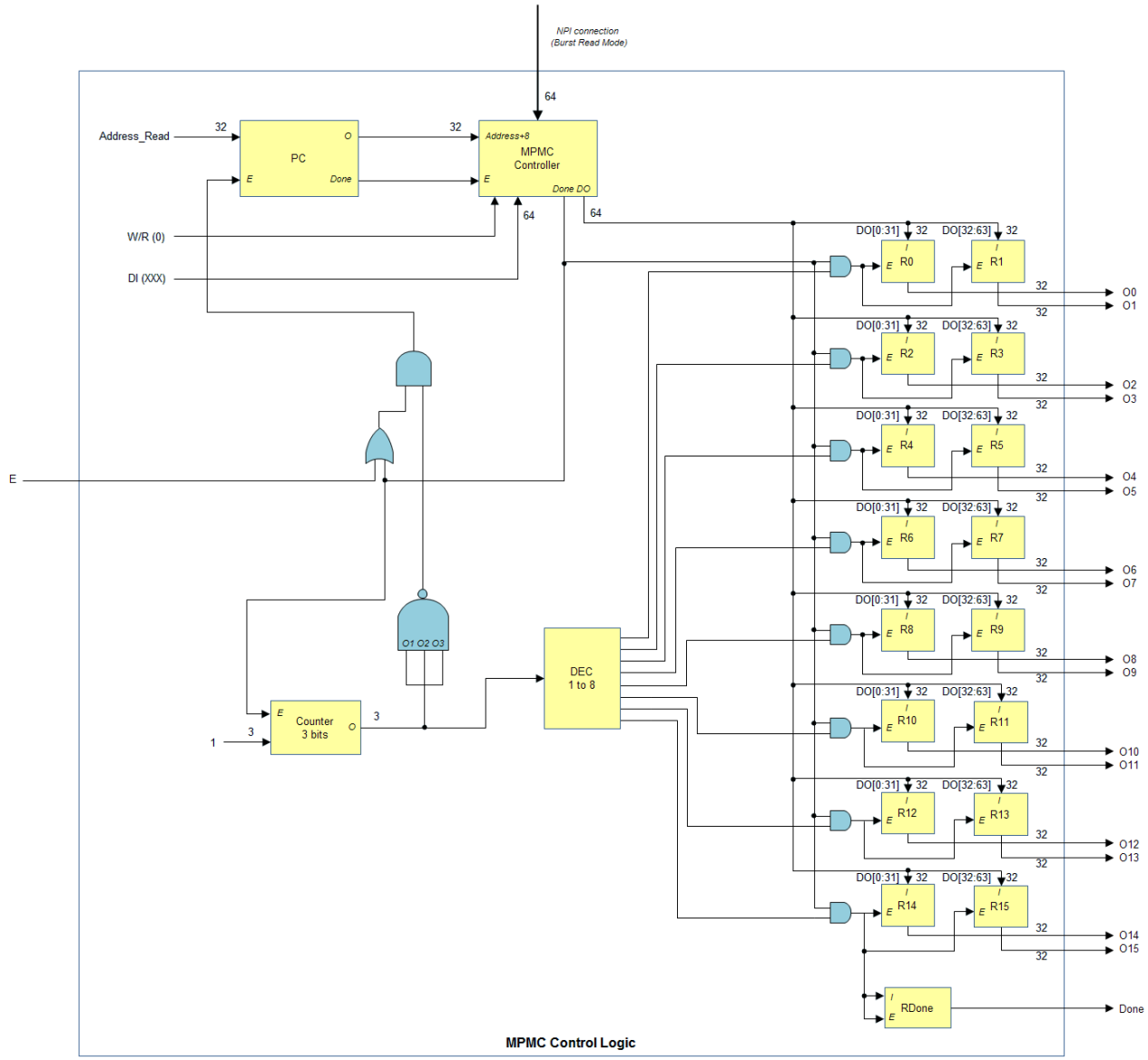

Figure 12. Diagram of the MPMC Control Logic module, responsible for obtaining the 8 values from memory for each port using the Burst Read mode.

2.2.1.1. The MPMC Controller.

The MPMC Controller read logic has been modified, since the Burst Read mode has been implemented as state machine (Figure 13). In the case of writes, this action is performed in the manner described in Section 2.1.1.1.

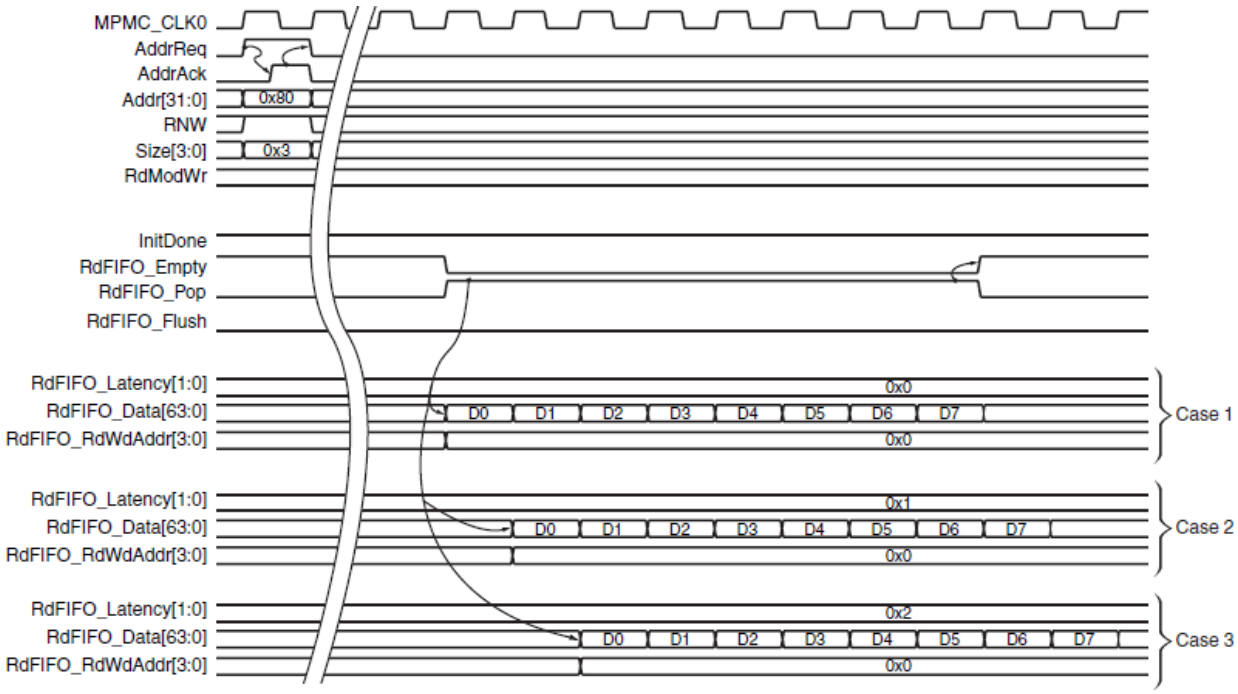

Figure 13: MPMC module control signals a read of a block of 64-bit words from memory, using the Burst Read mode.

3. Results.

All the circuits were designed, implemented, and executed on an ML507 Evaluation and Development Platform (which contains a Virtex-5 XC5VFX70T FPGA) using the Xilinx Platform Studio (XPS) 12.2 programming environment. Although newer versions of the Virtex FPGAs are now commercially available, the Virtex-5 version was used in order to guarantee a complete adaptation to the SpaceCube system. To do this, it was necessary to develop the hardware and software components separately:

- Hardware development: To develop the circuits that will be programmed in the FPGA, a complete system composed of a PowerPC processor and the peripherals was implemented, as shown in Figure 1. This design also includes a 64-kilobyte BRAM memory dedicated to the processor to store temporal data and the application instructions, along with a bus arbiter to handle communications. The two circuits presented were attached as different peripherals called Multi-spectral Distance Calculator (MEDiC), including registers (*slv_reg*) to store the input and output communications (to and from the PowerPC processor).

- Software development: A simple application that runs in the PowerPC processor at a clock rate of 400 MHz. This application randomly generates two 64-component (*A* and *B*) arrays and store them in the DDR memory, so that the peripherals can access them. Since the spectrum length is 64, and the developed hardware system processes up to 32 spectrum values at the same time, it is necessary to perform two iterations.

It was first necessary to check the critical path times between the Configurable Logic Blocks (CLBs) resulting from the *synthesis* and *routing* phases carried out by XPS. By using the additional Xilinx Timing Analysis tool, we found that the maximum path delay between CLBs was 7.537 ns, which allows a maximum theoretical frequency of 132.679 MHz. This means that the developed circuit will work correctly at clock speeds up to 132 MHz, and may present strange behavior and incorrect results if higher frequencies are used. Though the embedded PowerPC processor executes

at 400 MHz, the maximum frequency allowed by the clock of the PLBv46 bus (which regulates the peripherals attached to it) is 100 MHz. Therefore, the MEDiC system can operate at the full speed of the host bus with no latency issues.

After that, the hardware system was slightly modified in order to analyze the execution times of each of the proposed methodologies. A component called *counter* was added to accurately calculate the number of clock cycles spent by the modules of interest at a hardware level. This component has two inputs, *Start* and *Stop*, that deal with activating and deactivating, respectively, the counter and control the number of clock cycles that it executes. Adjusting these inputs, and the signals implemented in the proposed system, we measured the execution times for two kinds of components using two different counters, whose location within the system is shown in Figure 14:

- Counter 1 measures the time used for communications (reads of both complete spectra from the DDR2 memory). The *Start* signal is activated after the processor indicates the system to start working (Enable register), whereas the *Stop* signal is activated after all the reads from memory have been performed (output of the central AND gate). The result of this execution time calculation is stored in the CC_Comm register.

- Counter 2 measures the time used to calculate the multi-spectral Euclidean distance (Parallel MEDiC module). The Start signal is activated after all the reads from memory have been performed (output of the central AND gate), whereas the Stop signal is activated after the execution of the Parallel MEDiC module has finished. The result of this execution time calculation is stored in the CC_Ex register.

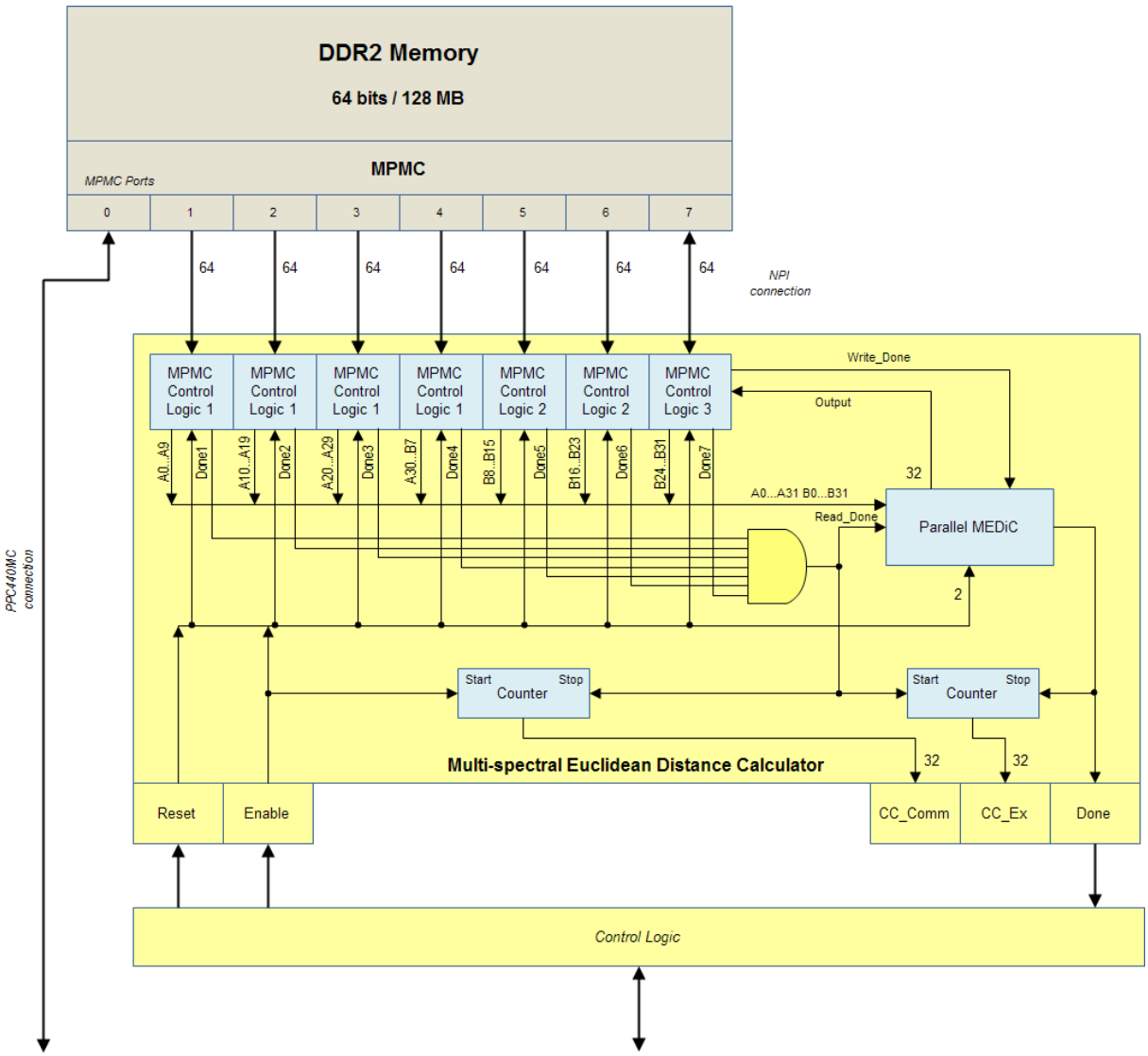

Figure 14: Diagram of the MEDiC peripheral with the execution time counters added.

Despite Figure 14 shows the addition of the counters for the first proposed system, they were also included for the second proposed system according to the inputs and outputs of the four MPMC Control Logic ports and of the corresponding Parallel MEDiC module.

To analyze the performance of the hardware circuits versus the software application, the multi-spectral Euclidean distance of the two arrays (spectra) was performed, and the execution times were obtained by the PowerPC from the CC_Comm and CC_Ex registers. The sum of the values in both registers will represent the total execution time of each proposed system.

These execution times were compared to two different versions of multi-spectral Euclidean distance calculations, also implemented and executed in the same Xilinx Virtex-5 boards:

- A software version of the multi-spectral Euclidean distance calculation between both spectra through 64 iterations of the following C statement: $E\_i = (A[i]-B[i])*(A[i]-B[i])$.

- A previous non-optimized hardware design, where the spectrum values were directly sent by the PowerPC processor to the MEDiC peripheral through the PLBv46 central bus and additional communication registers, using no intermediate memory. Also, the SQRT submodule used by this Parallel MEDiC module is not optimized; it consumes 17 cc instead

of 10 cc.

These execution times for all the proposed systems, including the software and non-optimized versions detailed above, are shown in Table 1.

	Software	**Hardware (Non-optimized)**	**Hardware (Design 1)**	**Hardware (Design 2)**
Execution time	6545 cc	4421 cc	612 cc	168 cc
Speed-up	6545 / 6545 = 1X	6545 / 4421 = 1.48X	6545 / 612 = 10.69X	6545 / 168 =38.96X

Table 1. Execution times for the calculation of the multi-spectral Euclidean distance calculations using the software and hardware versions.

Table 1 shows that the speed-up obtained for the non-optimized version of the designs is near to 1.5X, a low value that makes one consider an easier to program software version instead of a hardware version. Of course, this time consumption is due to non-optimized SQRT calculations and processor-to-peripheral communication times by means of the central PLBv46 bus. On the other hand, better results are achieved when using the shared DDR2 memory for processor-to-peripheral communications. The speed-up obtained for the first design of the multi-spectral Euclidean calculation operation is about 11x, while the second design manages a 40x speedup. In the second design, only four port connections were used, with the Burst Read mode enabled. Block reads from memory by using the Burst Read mode reduces the global computation time by 4x compared to the other the designs. Enabling this mode is crucial to designs where memory accesses are required.

These results show that the adaptation of basic, highly-used computing operations can improve the global performance and reduce computation time by 40x in spaceborne methodologies, when processing hardware devices such as SpaceCube are considered. Furthermore, since computation is offloaded from the main processor, it can perform other tasks until the result from external sources is obtained, which further improves the global performance of the spaceborne system. Also, since the system is reconfigurable, designers can improve or update the hardware modules in the FPGA, or even create new ones, while the satellite is already operating in space. Satellites using the SpaceCube platform could plausibly be totally reconfigured and adapted to new missions, re-utilizing, in this way, the satellite for new or adapted purposes once its primary mission is complete.

In addition, it is also interesting to analyze the clock cycles spent by the communications module and Parallel MEDiC module individually. This is possible with the values stored in the CC_Comm and CC_Exec registers when the execution finishes. These values are shown in Table 2.

	Hardware (Non-optimized)	**Hardware (Design 1)**	**Hardware (Design 2)**
CC Execution	26 cc	19 cc	19 cc
CC Communications	4395 cc	593 cc	149 cc
% Communications	99.41 %	96.9 %	88.69 %

Table 2. Clock cycles spent by the communications and the Parallel MEDiC module individually.

Table 2 shows that the main problems when using the PLBv46 central bus as the communication device are the bottlenecks present in the device and the slow speed when transferring data. Apart from causing high communication latency, they represent nearly 99% of the total communication time. Using a shared DDR2 memory as communication device between processor and peripheral,

with advanced transferring data techniques like NPI connections, decreases the time percentage dedicated to communications to nearly 97% for the first design. Moreover, if Burst read and write mode is activated, data is transferred in blocks at high speed, reducing the communication time to 88.7%.

4. Conclusions.

Methodologies that require satellite data are becoming more and more complex every day. It is necessary to design and implement new software and hardware algorithms to carry out these techniques as efficiently as possible. This need is more apparent when real-time on-board processing is required to properly perform all the tasks demanded by the methodology used. In this work, an operation executed commonly in satellite on-board processing algorithms, the Euclidean distance between two spectra, has been developed for a reconfigurable system. For that, advanced techniques such as shared memories and double-word burst read transfers were used. This improves the global performance and frees the processor of work, so it can complete other tasks. Two hardware versions were developed: one that makes use of seven memory ports using double-word transfers in normal mode, and a second that makes use of four memory ports using double-word transfers in read burst mode. Both circuits were executed and tested in a Virtex-5 FPGA, the same used by the SpaceCube platform, to demonstrate the ability to execute the algorithm in a real spaceborne system. Experimental results show that carrying out this basic operation in a hardware circuit is more efficient than the analogous software implementation and other hardware implementations involving other kinds of processors, like peripheral communication devices, by a factor of roughly 40x.

Acknowledgments

The authors would like to thank David Petrick and Jacqueline LeMoigne for their help by reviewing the paper; and Gary Crum, Daniel Espinosa, and the rest of the NASA Goddard Space Flight Center SpaceCube team for their support when developing this work.

This work has been funded with support from Gobierno de Extremadura and Fondos FEDER, within the project *Computación Paralela Utilizando Plataformas Grid de Algoritmos Hiperespectrales y su Aplicación al Seguimiento y Control de Incendios*.

The image in Figure 1 is courtesy of NASA. The images in Figure 6 and 13 are courtesy of Xilinx. The rest of images are of our design.

References

[1] Oaida B, Mercury M (2011). “HyspIRI mission concept”, HyspIRI Workshop, Washington DC, USA.

[2] McCarthy K, Stocklin F, Geldzahler B, Friedman D, Celeste P (2010). “NASA's evolution to Ka-band space communications for near-Earth spacecraft”, SpaceOps, Huntsville, USA.

[3] Aschbacher J (2010). “The GMES programme: Implementation of the space component and services”, Space Downstream Services, Tallinn, Estonia.

[4] Koga R, Crain WR, Crawford KB, Hansel SJ, Pinkerton SD, Tsubota TK (1992). “The impact of

ASIC devices on the SEU vulnerability of space-borne computers", IEEE Transaction on Nuclear Science 39(6), pp. 1685-1692.

[5] Wang JJ (2003). "Radiation effects in FPGAs", 9th Workshop on Electronics for LHC Experiments, pp. 34-43, Amsterdam, The Netherlands.

[6] Graham PS, Caffrey MP, Los Alamos National Laboratory, Wirthlin MJ, Johnson E, Rollins NH (2003). "Reconfigurable computing in Space: From current technology to reconfigurable systems-on-a-chip", IEEE Aerospace Conference, Big Sky, USA.

[7] Flatley T (2010). "Advanced hybrid on-board science data processor - SpaceCube 2.0", Earth Science Technology Forum, Arlington, USA.

[8] Xilinx official site: http://www.xilinx.com

[9] NASA Materials International Space Station Experiment - 7 (MISSE-7) official site: http://www.nasa.gov/mission_pages/station/research/experiments/MISSE-7.html

[10] Xilinx (2009). "Virtex-5 family overview".

[11] Marshall JR, Berger RW (2003). "Advancing reconfigurable processing subsystems in spaceborne applications", IEEE Aerospace Conference, Big Sky, USA.

[12] Gualtieri JA, Tilton JC (2002). "Hierarchical segmentation of hyperspectral data", Proceedings of the 2002 AVIRIS Earth Science and Applications Workshop, Pasadena, USA.

[13] Tilton JC (2003). "Hierarchical image segmentation", Journal of Space Communications 3.

[14] Xilinx (2010). "LogiCORE IP Processor Local Bus (PLB) v4.6 (v1.05a)".

[15] Xilinx (2011). "LogiCORE IP Multi-Port Memory Controller (MPMC) (v6.03.a)".

[16] Andraka R (1998). "A survey of CORDIC algorithms for FPGA based computers", International Symposium on Field Programmable Gate Arrays: 191-200, Monterey, USA.

[17] Xilinx (2011), "LogiCORE IP CORDIC v4.0".