AN ARCHITECTURE FOR REAL-TIME INTERACTION BETWEEN AN ANALOG RADAR WITH A PERSONAL COMPUTER

Daniel R. Llamocca-Obregón — *llamocca.dr@pucp.edu.pe*
Pedro M. Crisóstomo-Romero — *crisostomo.pm@pucp.edu.pe*
Grupo de Procesamiento Digital de Señales e Imágenes - Pontificia Universidad Católica del Perú
Av. Universitaria s/n Cuadra 18 - Lima 32, Perú. Telf.: +511-6262000 Anexo 4681

ABSTRACT

This work presents an architecture that acquires a set of digitalized signals from an analog radar, performs some processing, send packets of data through a USB interface, and finally, by software, displays the data sent by an analog radar. In this way, the analog radar is converted into a fully operative digital radar at a very small cost. Since the architecture does not require extensive hardware, it has been targeted to a CPLD, and the maximum frequency of operation obtained is 74.23 MHz.

1. INTRODUCTION

The low-cost digitalization of analog radars effectively transforms old fully operative analog radars into new digital ones. In the present work, the digitalization process consists in: the acquisition and processing of radar signals, the sending of packets of data to a PC thru a USB interface, and the development of a software interface in LINUX to display the results. The hardware uses a large number of ports, thus microcontrollers are not a good option, and a specific hardware is preferred. Besides, the use of specific hardware allows a faster acquisition and processing of the radar signals. The architecture uses few hardware resources, and fits easily into a CPLD. However, the USB interface does require extensive hardware resources. This problem is solved by using a board provided by ALTERA®: The MAX II Development Kit, which includes a MAX II EPM1270F256C5 CPLD and a USB chip (FT245BM from FTDI), with which the hardware has to interact.

The rest of this work is organized as follows. Section 2 describes the requirements of the system. Section 3 provides a detailed description of the architecture. Finally, conclusions and recommendations are given.

2. REQUIREMENTS OF THE SYSTEM

Fig. 1 shows the entire system. The dotted rectangle corresponds to the hardware described in this work.

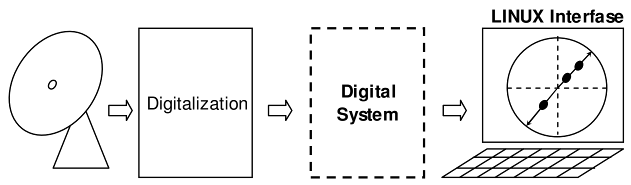

Figure 1. Complete digitalization system

Fig. 2 shows the interface designed, which is part of the digital system, along with its inputs and outputs:

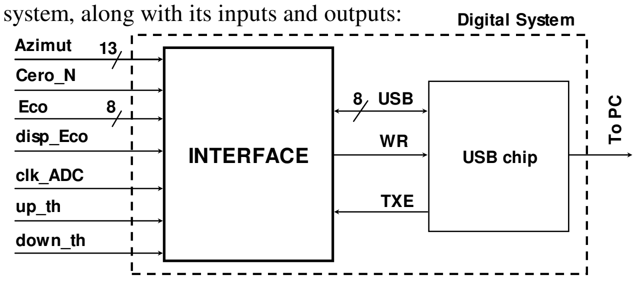

Figure 2

The angular position or azimuth of the radar, quantized in 1024 positions, is given by the *'Azimut'* signal (represented in BCD with 13 bits). If *'Cero_N'* is '1', it indicates that the reference for all the angular positions must be the geographic north, otherwise a manual adjustment must be done to obtain the reference, and it must be indicated via software. At each azimuth, it is possible that one or more objects appear.

A leading edge of *'disp_Eco'* indicates that a new *'Azimut'* is present (which must be captured along with *'Cero_N'*). Until a new leading edge of *'disp_Eco'* occurs (after aprox. 4 ms, or it automatically triggers after 10ms), all the objects spotted will have the same angular position. The group of *'Eco'* signals (captured in the falling edge of *'clk_ADC'*, which happens at a rate of 4 MHz) is used to identify objects: a group of *'Eco'* signals forms an object when there are a minimum number of contiguous *'Eco'* samples (e.g. 5 samples) above a threshold (which is controlled by the signals *'up_th'* and *'down_th'*). Fig. 3 shows a pattern for 2 objects with the same angular position:

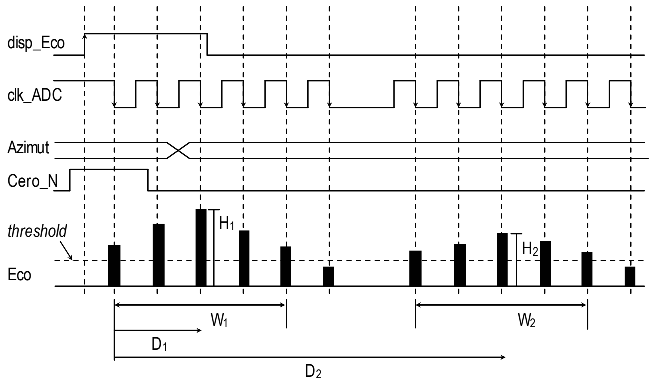

Figure 3. Pattern for 2 objects with the same Azimuth

For every object, we must obtain the distance of the object from the radar (D), the width of the object (W), and the intensity of the signal that comes back from the object (H). The distance (D) is calculated as the sample number of *'Eco'* for which *'Eco'* is the largest. The width (W) will be the number of samples above the threshold that forms an object. The intensity (H) will be the maximum value of *'Eco'* for a particular object. In Fig. 3, two objects appear and their respective distances, widths and intensities are shown.

For displaying, every azimuth is shown along with its objects. Fig. 4 shows 5 azimuths with its objects. The width, intensity and distance of each object are proportional to W, H and D respectively. For example, the azimuth indicated by the red line, along with its two objects, represents the pattern seen in Figure 3.

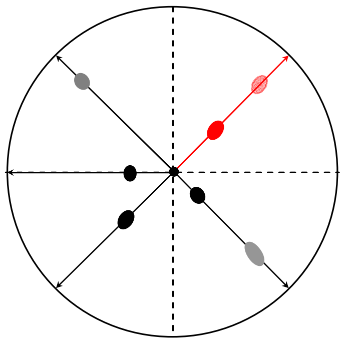

Figure 4. Displaying the results

For each object to be displayed correctly, we must send a packet thru the USB interface, containing the azimuth (A), the distance (D), the intensity (H) and the width (W) of every object. Recall that there can be more than one object for the same azimuth. A packet is sent when at least one object has been spotted or when no object has been spotted and a new leading edge of *'disp_Eco'* occurs. The packet contains 8 bytes, 1 byte is sent at a time, and there is a delay of 1 us for each byte, and is shown in Fig. 5:

FE	C	A(LSB)	A(MSB)	D(LSB)	D(MSB)	H	W

Figure 5. Format of a data packet sent to the USB chip

The headword 'FE' indicates the beginning of a frame, and there is also a counter (C: 0 to 255), which is useful for the software to know whether or not a frame is lost.

3. ARCHITECTURE IMPLEMENTED

The architecture of the interface shown in Fig. 2 is depicted. All the hardware is synchronized by a global clock signal of 66 MHz (which is not shown). It contains two main blocks. The block *'fsm_pri'* process all the incoming data and sent them in packets to the second block, called *'escr_USB'*, which receives the data from *'fsm_pri'* and controls the timing of the signals, according with the timing diagram of the USB chip, so it can correctly send data to the PC.

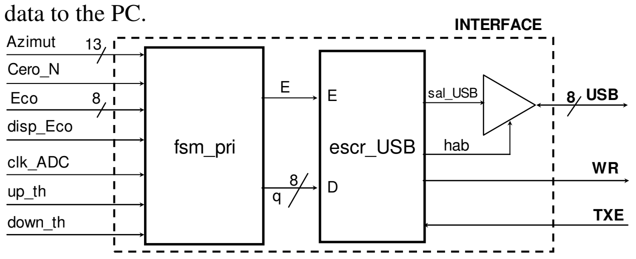

Figure 6. Block Diagram of the architecture.

3.1 Block *'fsm_pri'*

This block process the incoming data, and send it into packets. The internal architecture is depicted in Fig. 7. The internal signals are read or generated by the FSM shown in the figure.

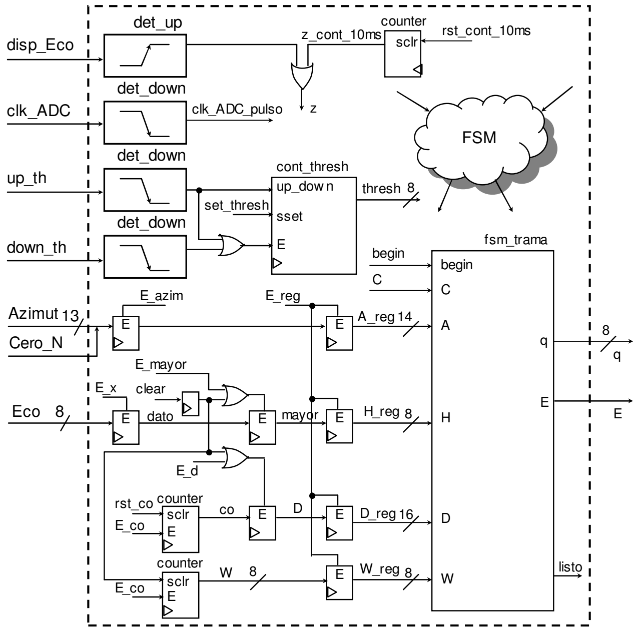

Figure 7. Internal architecture for *'fsm_pri'*

Note that 'Cero_N' is attached to the MSB part of the Azimuth (A). To obtain H, each new 'Eco' is compared with the largest previous value of Eco. We obtain W by counting the successive samples of 'Eco' above the threshold (initially 05). To be a valid object, W must be at least 5. To obtain D, we count the samples and store the count that produces the maximum value of 'Eco' for each object. The FSM for this block is depicted in Fig. 9.

The sub-block *'fsm_trama'* consists on a FSM (shown in Fig. 8) that receives A, W, D and H, and forms the packets as stated in Fig. 5. In Fig. 8, note that there is a delay of 1 us between every byte sent (the delay is controlled by the counter 0 → 65, note that before sending every byte the value *'cuenta'* must be 0).

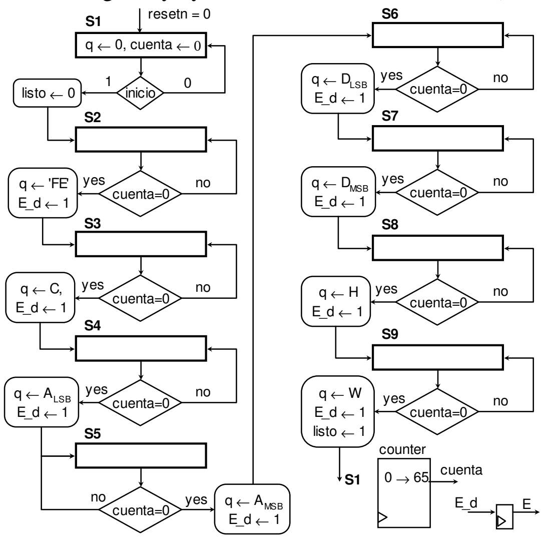

Figure 8. FSM for *'fsm_trama'*

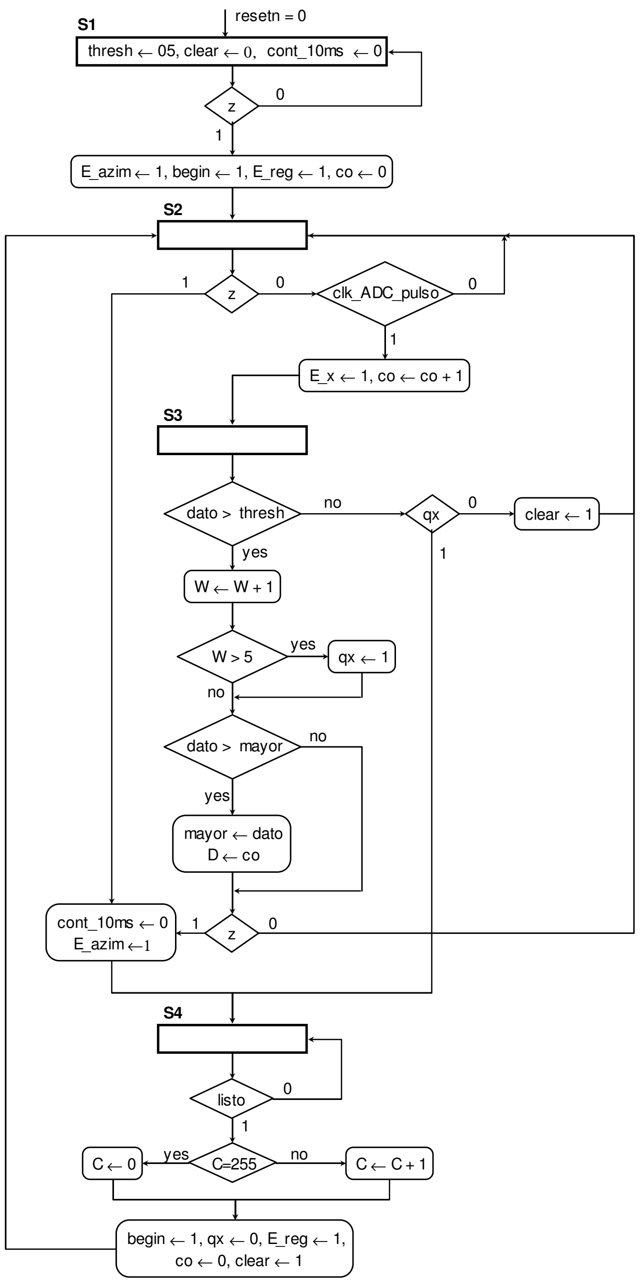

Figure 9. FSM of *'fsm_pri'*

3.2. Block *'escr_USB'*

This block (shown in Fig. 6) generates the set of signals to correctly communicate with the USB chip, so that the USB chip captures the correct data to be sent to the PC.

The inputs are: 'D', of 8 bits, and 'E', which indicates that the input on 'D' is valid and must be sent to the USB chip. Fig. 10 shows the timing diagram of the write cycle of the USB chip [2]. The input signal to the USB chip is 'TXE' and the output signals sent to the USB chip are 'WR' and 'USB'.

The block *'escr_USB'* will read the signal 'TXE', and will generate the signal 'WR'. It will also generate the signals 'hab' and 'sal_USB' that are the input signals of the tri-state buffer, which will generate the signal 'USB'.

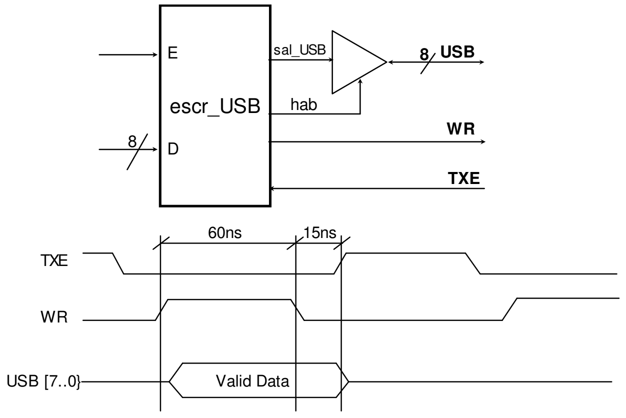

Figure 10. Timing diagram of the USB chip

The following is the strategy to design the FSM that implements the timing diagram of the USB chip:

a) Wait until TXE = '0'.
b) If TXE = '0', then we have to set WR = '1' and put the Valid Data on the Bus (hab = '1' and sal_USB 🡐 D)
c) Wait 60 ns (approx. 4 clock cycles) and then set WR = '0'.
d) Wait 15 ns (approx. 1 clock cycle) and retire the Valid Data from the bus (hab = '0').
e) Wait until TXE = '1'.
f) Return to step 'a'.

Figure 11 shows the FSM that corresponds to the block 'escr_USB'. Note that the FSM considers the signal 'E': only if 'E' is asserted, the FSM put a valid data on the bus. Also, Note that the output 'sal_USB' is registered and takes the value of 'D' whenever the signal 'ED' is asserted.

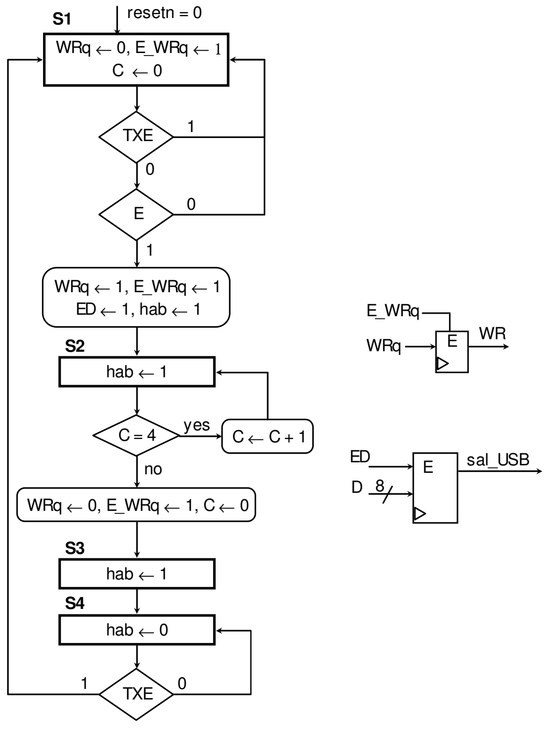

Figure 11. FSM for the 'escr_USB' block

3.3. Results of CPLD implementation

Table 1 shows the resource effort and maximum frequency of the iterative architecture that implements the *ln* function.

CPLD:	**LE**	**f_{max} MHz)**
MAX II EPM1270F256C5	348	74.23

Table 1. Final Results

4. CONCLUSIONS

- The digitalization of an analog radar has proved to be amenable for our CPLD implementation, as the clock rate and resource effort indicates.
- The large number of pins and the need of a USB interfase make it very difficult to use a microprocessor. The USB chip provided by the MAXII Development Kit makes it very easy to communicate the CPLD with the PC.

5. REFERENCES

[1] Brown & Vranesic. Fundamentals of Digital Logic with VHDL Design, McGraw Hill, 2000

[2] FT245BM USB FIFO (USB - Parallel) I.C. data sheet, Future Technology Devices (www.ftdichip.com), 2005.

[3] ALTERA CORPORATION, “MAXII Development Board Data Sheet”, October 2004.