Automatic Frequency Controller for Aging Effects Mitigation in FPGA-based Designs

C. Leong\(^1\), J. Semião\(^1,3\), M.B. Santos\(^2,3,4\), I.C. Teixeira\(^2,3\), J.P. Teixeira\(^2,3\)

\(^1\) University of Algarve, Campus da Penha, 8005-139 Faro, Portugal, \{a15297, jsemiao\}@ualg.pt
\(^2\) IST, Lisboa, Portugal, \{marcelino.santos, isabel.teixeira, paulo.teixeira\}@ist.utl.pt
\(^3\) INESC-ID Lisboa, R. Alves Redol 9, 3º, 1000-029 Lisboa, Portugal
\(^4\) Silicongate, R. Alves Redol 9, 1000-029 Lisboa, Portugal

Abstract – In this report, a methodology for aging effects mitigation in FPGA-based designs is described. On-chip delay sensors (built with standard FPGA resources) are used to detect aging effects and an automatic frequency controller (AFC) is used for aging effects mitigation, driving the modulation of the system clock frequency. Two Local Aging Sensor (AS) architectures are presented, using single and dual clock signals. The focus of the report is the design and implementation of the automatic frequency controller, demonstrated in a case study with a single clock AS. Main applications are in the area of long-term operation systems, particularly, in safety critical systems.

I - INTRODUCTION

As IC technologies move from one node technology to the following one, sensitivity to variations increase, as well as reliability concerns [1]. The existing variability quest for the use of wider safety-margins, associated with design corners that consider worst case PVT (Process, supply Voltage and Temperature) variations. Nevertheless, energy efficiency without performance degradation is demanded. Hence, the impact of variability becomes a major concern.

FPGA (Field Programmable Gate Array) vendors provide conservative specifications and tools that use (in simulation) pessimistic worst-case PVT conditions [6]. Aging effects are not usually taken into account. As a consequence, maximum allowable (by design) system clock frequencies (\(f_{clk}\)) are significantly lower than the ones a typical component can deliver [5]. Dually, in physical devices a given performance (\(f_{clk}\)), may be achieved with lower power consumption, by down grading the core supply voltage, \(V_{DD}\).

On the other hand, for products that must operate for long periods of time, performance degradation, over the years, must be under observation. Device reliability (or product lifetime) may be improved, if the system could adapt either \(f_{clk}\) or core \(V_{DD}\), according to its own graceful degradation. Therefore, performance monitoring can be rewarding to fully take advantage of the available performance, at production stage, but also to guarantee safe operation, during product lifetime.

Therefore, aging effects must also be taken into consideration since they can degrade system performance and ultimately cause system malfunction. This is particularly serious in critical application systems characterized by not allowing tolerance to failures. Sensors are used in these systems for detecting aging effects on system performance and behaviour.

Aging effects, namely due to NBTI (Negative Bias Thermal Instability), have been studied and an array of on-chip sensors that locally monitor system performance have been developed in the context of ASIC design (see, namely [2][3][5][13][14][15][16][17]).

On the contrary, limited work has been carried out for on-chip sensor design for FPGA-based systems [4][6][18][10][11][12]. One of the difficulties is that, in FPGA, each time a sensor cell is instantiated, different resource allocation and performance occurs. This is different from what happens in ASIC (Application Specific Integrated Circuit) design, where cell instantiation presents always the same layout and performance.
However, reconfigurable devices, like FPGA, using state-of-the-art technology, are increasingly being used to replace ASICs especially in low-volume applications. Hence, to develop novel solutions to monitor circuit performance and application-dependent circuit aging in FPGA-based designs [4] becomes mandatory.

In order to achieve this purpose, an aging monitoring methodology based on delay sensing for FPGA-based designs, using standard FPGA resources (hardware and vendor tools) is under development [24]. Different on-chip FPGA delay sensors have been proposed in [10][11][12] and [24]. In [10] the proof of concept of using sensors for detecting aging effects has been presented, with a single sensor insertion and a simple CUT (Circuit Under Test). In [24], the methodology has been updated to allow multi-sensor insertion using different criteria for sensor insertion and to avoid the need of the design freeze (slice and routing locking) of the “clean” design (design version before sensor insertion).

In order to move towards an adaptive system design, after aging effects are monitored, such data must be reused to trigger the appropriate action to preserve the safety margin in device operation, as far as performance is concerned. In Fig. 1, the overall scheme of the proposed methodology for adaptive system design of FPGA-based systems is shown. At present, short term effects are considered. In order to implement the methodology, there is the need to develop an on-chip automatic frequency controller (AFC), using standard FPGA resources. The AFC will control the system DCM (Digital Clock Manager) to modulate f_clk, basically down grading it, when the component is aging. The purpose of this report is to propose the architecture and to design and implement a built-in AFC to mitigate the aging effects.

![Fig. 1 – Methodology for adaptive FPGA-based system design](image)

This report is organized as follows. In section II, the aging monitoring and mitigation methodology is presented. Sensors used for delay sensing are described in Section III. The automatic frequency controller is described in section IV. The configured testbench (case study) as well as simulation and experimental results are presented in Section V. Conclusions are summarized in section VI.

**II. AGING MONITORING AND MITIGATION METHODOLOGY**

The methodology relies on on-line performance monitoring, at production or during product lifetime (aging monitoring). Two types of sensing devices can be built in the FPGA tissue, namely, **Global** [6] and **Local sensors**. Global sensors are basically delay sensors which are independent of any CUT configuration; they use unallocated FPGA resources and act as coarse-grain sensors, unfitted for safety-critical applications. Local sensors are application-dependent and monitor for a given configuration the CUT critical paths. In this work, only Local sensors are considered. In recent developments, the methodology targets aging effects mitigation in FPGA-based designs (Fig. 1). This constitutes a significant improvement, since, as mentioned before, for products that must operate for long periods of time performance monitoring and mitigation of aging effects need to be carried out to guarantee safe operation, during product lifetime.

In the first step a “clean” design is generated to configure CUT functionality. Vendor tools estimate CUT performance (f_clk_sim) using worst-case PVT conditions, and identify the signal CPs (Critical Paths). The predicted time response of the largest signal delay under observation (the longest CP) is used to identify the registers at which inputs terminate CPs; these are assigned as CMEs (Critical Memory Elements). CMEs with a high switching activity will likely activate their CPs. CMEs with low switching activity (mainly with ‘0’ at their
inputs) will age faster, if NBTI is the dominant aging effect. Aging sensors (AS) are to be inserted in key locations monitoring CMEs inputs (Fig. 2). The data collected by the AS is used to inform the AFC, in order to correct $f_{clk}$ and thus to mitigate the aging effects.

For aging monitoring, aging sensors are inserted to predict delay errors, before they start occurring. Such predictive error detection guarantees safe operation (vital to safety-critical systems), while action can be taken to ease the time margins, slightly decreasing the system clock frequency. This approach is different from the typical one, (see e.g., the RAZOR technique [22]), in which error detection and correction (EDC) is used. In our case, the time safety margin, $\Phi$, is user’s defined.

In order to monitor the propagation delay times of the selected CPs, two solutions exist that may shape the AS architecture. Either we capture the CME input signal (Si signal) sooner than the CME capture time, or we delay Si, in order to capture the CP response simultaneously with the CME.

In the first solution, a second signal clock must be generated on-chip, with the same clock frequency but with a phase shift, $\Phi$, in advance (relatively to Clk). In the second solution, no additional signal clock is required; instead, a delay element (DE) is used to delay Si by the same amount of time, $\Phi$. The two solutions are discussed in more detail in Section III, where two correspondent AS architectures are presented. In FPGA devices (we consider Xilinx™ devices, Virtex5 or Spartan6 families), clock signals generation and management require the use of on-chip DCMs.

For aging effects mitigation the following procedure is implemented. When safety margin violation is detected by the sensors, mitigation can be achieved, either by reducing the working frequency ($f_{clk}$), or by increasing the core $V_{DD}$ voltage, since both procedures guarantee that performance is maintained and malfunction is avoided.

At the present, aging effects are mitigated by modifying the working frequency. For this, an automatic frequency controller (AFC) is required. The automatic frequency controller provides the new clock frequency both to the CUT and to the aging sensors.

As no design freeze is required in the proposed methodology [24], a new CUT configuration with multi-sensor insertion is generated. This new configuration may modify the critical paths location and their propagation delays values. Therefore, non-critical paths may now be monitored, while new CPs may not be monitored. Fortunately, in a good design, many critical and near-critical paths converge at few registers, selected as CMEs. Moreover, if a “comfortable” safety margin is used, but still allowing significant performance or power gains, it has been observed [24], by using the proposed methodology, that sensor insertion is reliable and performance monitoring is effective. Typically, we have observed that the majority of the sensors still operate monitoring CPs.
Main actors in this methodology are the *aging sensor* and the structures that generate the sensors control signals, particularly, the *clock frequency* (Fig. 2). Moreover, the aging sensor architecture must be designed with standard FPGA resources, to be automatically inserted and compatible with FPGA’s registers.

In Fig. 2, System clock, Clk, is delivered through a *system DCM*, which will also be used to modulate the clock frequency, if required during product lifetime. A key issue in sensor insertion is the tight control of $\Phi$, since it depends on the interconnections delays $td1$ and $td2$ associated with the corresponding routing process (Fig. 2). This is particularly relevant for multi-sensor insertion, since a given logic implementation depends on its location in the FPGA. In the following sections, two *aging sensor* architectures and the *automatic frequency controller* will be described.

### III. AGING SENSOR WITH SHIFTED CLOCK

As mentioned before, the aging sensor architecture is designed to be automatically inserted and compatible with FPGA’s registers. Moreover, it is compatible with CME implementing D, CE and SR flip-flops. Typically, several Critical (or nearly critical) Paths (CPi) converge to a given CME.

Two AS architectures have been proposed. These 2 architectures correspond to the 2 solutions mentioned in Section II. Their main characteristics are summarized in Table 1.

<table>
<thead>
<tr>
<th>Xilinx FPGAs plus ISE tools</th>
<th>Aging Sensors (AS) architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>v2 (DUAL CLOCK)</td>
</tr>
<tr>
<td>outputs</td>
<td>v3 (SINGLE CLOCK)</td>
</tr>
<tr>
<td>single</td>
<td>X</td>
</tr>
<tr>
<td>multiple</td>
<td>X</td>
</tr>
<tr>
<td>Sensor DCM &amp; its controller</td>
<td>1 (for all AS)</td>
</tr>
<tr>
<td>DCM primitives</td>
<td>DCM_SP</td>
</tr>
<tr>
<td></td>
<td>DCM_CLKGEN</td>
</tr>
</tbody>
</table>

Table 1 – Proposed Aging Sensors (AS) architectures [24][25].

Previous results in developing AS for FPGA-based designs are referred as v1 (version 1), and they basically correspond to the research work performed with the University of Vigo, Spain, and PUCRS, Porto Alegre, Brazil [26].

Both v1 and v2 AS architectures correspond to the first *delay monitoring* solution: they are **Dual Clock sensors**, as a second clock signal, Clk_d, has to be generated on-chip. This requires at least one additional DCM to program the safety margin, $\Phi$. The development of such sensor, and the corresponding methodology for multi-sensor insertion, is described in [24] (FPL’13).

A novel architecture using the second *delay monitoring* solution, a **Single Clock sensor** (v3), has been proposed in [25]. The v3 architecture does not require the sensor DCM, and is more flexible, as it may be implemented with single or multi-output configurations.

The sensor architecture is depicted in Fig. 3 for dual and single clock architectures. The Dual Clock aging sensor captures the CME input signal ($Si$). Capture is *made in advance*, since a new clock signal, Clk_d, generated by a *sensor DCM* (Fig. 3 (a)) has a negative phase shift, $\Phi$. TAP is an 8-bit word that modulates the clock phase, $\Phi$ (safety margin). If a $Si$ transition occurs within the timeframe defined by $\Phi$, OUT_AS goes to ‘1’, ‘predicting’ a logic error. The *sensor DCM* feeds all local sensors (one per CME), thus restricting design overhead. Instead, the Single Clock aging sensor captures $Si$ and delays it (through a DE (Delay Element) to generate the safety margin, $\Phi$. 

---

4
In these sensor architectures the fast carry chain (Fig. 2) is used to transfer Si to the sensor’s input and, thus, to bound $\Delta t_d = (td2 - td1)$, and the HBLK_NM attribute (with Xilinx™ ISE 14.1 version) is used to lock the flip flops in place. This attribute guarantees that associated components are allocated into the same slice. Using the fast carry chain, the sensor’s intrinsic delay is limited and very well controlled (simulation results show 0 and 10 ps for the carry chain (inter slice propagation delay only) plus 330 ps of associated logic in Virtex5, and 3, 82 and 123 ps, correspondent to a normal, mid clock region gap and cross clock region for the carry chain plus 245 ps of associate logic in Spartan6 devices. Fig. 4 shows the resource allocation of a given CME and the local part of its corresponding aging sensor.
The principle of operation of the proposed Single Clock sensor (v3) is based on the existence of a controlled safety-margin, preceding the clock active edge. However, *this safety margin is virtual*, as there is no explicit signal defining it. In fact, the CME data input is connected to a *delay element* (DE), implemented with the carry-chain structure, which creates a longer critical path that will be captured by an additional flip-flop (in the sensor). Therefore, the delay of this DE will define the safety-margin (virtual guard-band) for the sensor to work on. This is a relevant improvement, in comparison to the previous proposals, since it avoids clock domain crossing problems (when two clocks exist) [27]. Moreover, in the proposed architecture, the safety margin increases with aging; this favors aging monitoring.

During this safety-margin, referred also as guard-band interval \( T_g \), or guard time), the sensor is active and monitors the late transients of the data input of the CME. The time interval, \( T_g \), is defined for each circuit application by the end-user, with a pre-determined resolution (that depends on the operating frequency and on the target FPGA). If, during \( T_g \), a late transient in the CME input data is detected by the sensor, a signal flags unsafe operation. This means that the safety-margin has been violated (although the correct data is still being captured by the CME).

The sensor functionality is similar to the ASIC solution presented in [14]. The sensor architecture in the FPGA solution is depicted in Fig. 5. As shown, the monitored node is Si (CME input). D line is delayed by the carry chain and captured by AS_FF1_i (Aging Sensor) flip-flop.
In this sensor architecture, several sensor elements are considered. Each one (Core AS element) is associated with a given user’s defined guard-band, and a corresponding delay, $T_g$. The node sensing can be carried out by one, several or by all stages. Hence, the proposed sensor, using several delay elements, allows scalability, since it allows guard-band parameterization. A typical solution is to use one stage (typically, the same $T_g$ for all CME) that triggers an alarm when a failure is predicted, or two stages to allow automatic adjustment of frequencies and/or supply voltage. Hence, the existence of several stages allows a flexible scheme for frequency and/or voltage adjustment. Thus, the single clock AS has the attribute of multiple output ports. In this work, this v3 AS architecture is used in association with the novel AFC controller.

IV – AUTOMATIC FREQUENCY CONTROLLER (AFC)

In earlier versions of the aging sensing methodology, the system clock, Clk, was delivered through a System DCM, which was also used to modulate the clock frequency, if required. Such clock frequency variation was performed manually. For instance, in [24], $f_{\text{clk}}$ was first augmented (using the System DCM) until we reach the maximum allowable clock frequency ($f_{\text{clk, max}}$), for each Spartan6 board. According to a user’s defined safety margin, this was the clock frequency for which no AS would flag a guardband violation. Then, the System DCM was programmed to increase $f_{\text{clk}}$ and to identify when the AS started to flag an abnormal delay, until a real error (due to an erroneous logic value capture) was perceived. This was not used during product lifetime; instead, it was used to demonstrate that the embedded local sensors were really able to predict a timing violation that could endanger safe operation. In that experiment, as we used a dual clock AS (v2), the safety margin was manually adjusted by a Sensor DCM used to program the phase margin, $\Phi$, through the TAP signal. For this, a Xilinx™ DCM primitive, the DCM_SP primitive, was used [23].

Presently, an automatic frequency controller (AFC) is used to modulate the clock signal to be fed to the CUT and to the set of local AS embedded in the CUT. The AFC / System DCM block diagram is depicted in
Fig. 6. As it can be observed, the DCM controller modulates the clock frequency generated by the System DCM. For this, another Xilinx™ DCM primitive, the DCM_CLKGEN primitive, is used [23] (Fig. 7).

Fig. 6 – Automatic frequency controller (AFC) / system DCM architecture

In Fig. 8, it is shown the interface between the novel AFC module and the DCM_CLKGEN primitive. Annexes I and II contains the description of all the I/O variables. Having this information and the functionality of the AFC module, it is possible to define AFC Inputs and Outputs.

Figure 7 – Xilinx™ Spartan6 DCM primitives [23]
IV.1 CONTROLLER INPUT/OUTPUTS

For an implementation in Xilinx™ technology, the DCM Controller Inputs are as follows:

- **Clk** (Clk_in) – controller clock
- **Reset** (Sys_RST) – controller reset
- **DataValue** – generic 8 bit word
- **ProgM** – when high, controller will capture DataValue as the value for M (Multiplier)
- **ProgD** – when high, controller will capture DataValue as the value for D (Divider)
- **SetDCM** – a high pulse of at least 1 clock wide will initiate the DCM programming sequence with the captured values of M and D
- **SensorIn** – 2 bit sensor input to indicate how the circuit is performing with the current clock frequency (these data come from the AS_OUT signals)
- **AutoSeek** – when high, the controller will auto adjust the M and D values according to the SensorIn input values and will program the DCM at set intervals
- **DCM_ProgDone** – return bit from the DCM to indicate that the DCM has finished the last instruction
- **DCM_Status** – 2 bit status given by the DCM (currently not used)

The DCM Controller Outputs are as follows:

- **DCM_ProgData** – serial data stream to program the DCM
- **DCM_ProgEn** – enable bit to program the DCM
- **DCM_ProgClk** – DCM programing clock (currently not used, hard wired to Clk outside this controller)
- **ResetSensor** – reset all aging sensor and BIST results
- **RetM** – 8 bit word of the current M value
- **RetD** – 8 bit word of the current D value

IV.2 – DCM PROGRAMMING

The System DCM clock signal output (Clk) frequency is determined by:
\[ f_{\text{clk}} = \frac{(M + 1)}{(D + 1)} \times f_{\text{clkin}} \]

where \( f_{\text{clkin}} = 100\text{MHz} \) from the Atlys onboard crystal clock generator. Five Spartan6 (xc6slx45-2csg324c) FPGA boards were used to validate the proposed automatic clock frequency controller.

To program the DCM, a protocol must be followed. Figure 9 depicts such protocol, as suggested by [23].

**IV.3 – CONTROLLER MODES**

The frequency controller has two main operation modes: *manual programming* and *automatic programming*.

**Manual Programming:**

1) Set the M value by inputting it into the DataValue port and activate ProgM. This will store the M value within the controller, as it can be verified at RetM.
2) Set the D value by inputting it into the DataValue port and activate ProgD. This will store the D value within the controller, as it can be verified at RetD.
3) Put a high pulse of at least 1 clock wide into SetDCM. If DCM_ProgDone is high (indicating that the DCM has completed its last programming) then the controller will initiate the DCM programming cycle. It will activate DCM_ProgData and DCM_ProgEn, according to the DCM programming protocol. (If DCM_ProgDone is low while the pulse of SetDCM is applied, depending on conditions, the controller may ignore this command and it must be repeated)

**Automatic Programming:**

Automatic programming is active while AutoSeek is high. The search starts with the current stored M and D values and is divided in 3 phases:

1) **Coarse Grain Search:**
   a. Starts by reading the SensorIn values after a preset time, in order to give some time for the sensors to act in case the circuit has just powered up.
b. While SensorIn values remain the same after each successive preset time, the M and D values are adjusted according to Table 2 and the DCM reprogrammed.

<table>
<thead>
<tr>
<th>SensorIn</th>
<th>Current status</th>
<th>00</th>
<th>10</th>
<th>X1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial status</td>
<td></td>
<td>while M &lt; 255, M++</td>
<td>next phase</td>
<td>next phase</td>
</tr>
<tr>
<td>10</td>
<td>next phase</td>
<td>next phase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X1</td>
<td>next phase</td>
<td>next phase</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2 – M and D adjustment (coarse grain search). “Next phase” means moving to the fine grain search.

2) **Fine Grain Search:**
   a. At each preset time interval, the SensorIn values are read and M and D values are adjusted according to Table 3 and the DCM reprogrammed.

<table>
<thead>
<tr>
<th>SensorIn</th>
<th>00</th>
<th>while M &lt; 126 and D &lt; 126, M = M x 2 + 2 and D = D x 2 + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>else, next phase</td>
<td></td>
</tr>
<tr>
<td>X1</td>
<td>while M &lt; 126 and D &lt; 126, M = M x 2 + 2 and D = D x 2 + 1</td>
<td>else, next phase</td>
</tr>
</tbody>
</table>

Table 3 - M and D adjustment (fine grain search)

3) **Monitoring**
   a. At each preset time interval, the SensorIn values are read and only M value is adjusted (subtracted by 1) if SensorIn is X1 and the DCM reprogrammed.

**V – CASE STUDY**

The CUT is assumed to be a typical functionality used in safety-critical applications. The core functionality is a Data Acquisition (DAQ) module used as a part of the data processor of a DAQ FPGA design (PEM_Daq_Sort_Crystal module) for a medical imaging system used in Positron Emission Tomography (PET) - based mammography [19]. For fast data acquisition and processing, a large number of input channels are used and parallel processing is carried out. Hence, several DAQ modules may be configured in a single FPGA.

In our testbench, the DAQ uses *Triple Modular Redundancy* (TMR) architecture to identify SEU (Single Event Upsets) which may cause one out of the three modules to fail. In this context, the TMR is able to detect and correct the error, through the voting system. Consequently, a single *DAQ unit* includes 3 *DAQ modules*, as shown in Fig. 10. Moreover, in order to have a larger CUT, 9 DAQ units have been configured in the testbench described in [24], to be stimulated either by the same test data, or by various test data. In [25], only one DAQ unit is used in five Spartan6 and one Virtex5 boards. Each DAQ module uses 2 DSP. Hence, in [24] the 9x3=27 DAQ modules use 54 DSP, close to the limit (58) of the Spartan6 device. Virtex5 devices use a 65 nm IC technology, while Spartan6 devices are manufactured with a 45 nm IC technology.
Fig. 10 - DAQ unit block diagram, showing the TMR architecture

Fig. 11 - Configured testbench to perform off-line BIST (9 DAQ units, dual clock AS (v2), additional DCM [24]). If a single clock AS (v3) is used, no Sensor DCM is used, and Clk_d=Clk.

A BIST (Built-In Self-Test) architecture is also configured in the FPGA to allow easy generation of test stimuli, and to allow the analysis of test responses (fig. 11). The TPG (Test Pattern Generator) is built either with two 32-bit LFSR (Linear Feedback Shift Registers) with different seeds [24], or with two 20-bit counters [25]. In [24], from the 64 bit, a Selector (Fig. 10) generates 8 test sequences of 42 bit words to be inputted to the DAQ modules. The 8 test sequences are sets of $2^{21}=2097152$ test vectors. The sensor DCM (Fig. 11) shifts the phase of the sensors’ clock signal, Clk_d. The signature analyzer is a CRC (Cyclic Redundancy Checking) block. Flashing LEDs, in the board, signal the clock signals. Errors are latched and outputted to LEDs for visualization. Error or warning signals are AS_OUT (aging sensors flagging abnormal delays), TMR_Error (2 bit signal, corresponding to flag recoverable (TMR0) and unrecoverable errors (TMR1)) and CRC_Error (erroneous CUT response).

The DAQ module block diagram, in which 5 aging sensors are inserted, is depicted in Fig. 12. Each DAQ module has 2 multipliers to process the incoming data. Data are then feed to the normalizers for processing. Data from the normalizers are captured by registers before being sent to the controller. Based on the inputs, the controller will select the appropriate data to be passed to the outputs and generate the necessary control signals. Five sensors are added on top of the registers under monitoring for aging effects (CMEs). CME selection has been made using ISE tools, based on the clean design (i.e., without sensors). In [24], with 9 DAQ units, 135 sensors have been inserted. The resource overhead is shown in Table 4 [24].
As referred, aging “emulation” is performed by increasing $f_{clk}$ beyond $f_{clk\_sim}$. For the CUT under study in [24] (9 DAQ units, v2 AS), ISE predicts $f_{clk\_sim} = 166$ MHz in Spartan6. In [25] (1 DAQ unit, v3 AS) ISE estimates $f_{clk\_sim} = 272$ MHz in Virtex5. In Spartan6, for the same (1 DAQ unit, v3 AS) circuit, ISE predicts $f_{clk\_sim} = 170$ MHz.

The AFC modulates the clock frequency adjustment, as the component will age, and incrementally the AFC drives the DCM_CLKGEN module to decrease the system clock frequency. For the (9 DAQ units, v3 AS) circuit with AFC, ISE estimates $f_{clk\_sim} = 168$ MHz in Spartan6 devices.

V.1 – FPGA RESOURCES

Table 4 summarizes the resources used with the example test circuit one TMR DAQ (the CUT) with 15 single clock AS (v3) of 7 core AS elements each. In [25], in order to evaluate how the sensor delays scale, up to 11 Core AS elements were inserted in each sensor location (of those, only 6 for Virtex5 and 5 for Spartan6 are actually outputted to the boards available LEDs).

<table>
<thead>
<tr>
<th>Resource\config</th>
<th>clean</th>
<th>fully configurable</th>
<th>fix SensorIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF</td>
<td>8736</td>
<td>11030</td>
<td>+26,26%</td>
</tr>
<tr>
<td>LUT</td>
<td>7900</td>
<td>10329</td>
<td>+30,75%</td>
</tr>
<tr>
<td>Slice</td>
<td>3614</td>
<td>4376</td>
<td>+21,08%</td>
</tr>
<tr>
<td>BufG</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>DCM</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DSP</td>
<td>54</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>Tclk (ns)</td>
<td>5,931</td>
<td>5,938</td>
<td>+0,12%</td>
</tr>
</tbody>
</table>

Table 4 – Resource allocation for the circuit (9 DAQ units, v3 sensors) with the proposed AFC in Spartan6, TPG implemented with LFSRs.

V.2 – RESULTS

A ML550 board from Xilinx with a Virtex5 (xc5vlx50t-1ff1136c) and five Digilent’s ATLYS boards with Spartan6 (xc6slx45-2csg324c) FPGA devices were used to validate the proposed methodology and sensors architecture. The Virtex5 board allows core $V_{DD}$ (from 0.95V to 1.05V) and clock frequency variations, which allow us to additionally demonstrate the influence of $V_{DD}$ variations on CUT performance. However, this is out of the scope of this report.
When the device is powered up, the AFC starts at 100 MHz clock frequency (the frequency of the external clock, generated in the board, and start **multiplying** the frequency: $M=2$ (200 MHz), $M=3$ (300 MHz), etc. With this coarse grain search, the frequency is increased until the circuit fails (CRC). For instance, for the case study whose results are shown in Fig. 13, for $M=3$ the CUT fails (for all boards, the CRC flags a system error for frequencies between 246 and 233 MHz). According to the algorithm, the AFC starts driving a frequency **reduction**. For 200 MHz the CUT responds correctly, and for 300 MHz it fails. Hence, the AFC divides this interval in equal intervals, and generates 250 MHz. For this frequency, the CUT fails; hence the AFC divides again the clock frequency interval in the middle, and generates 225 MHz. Experimental results shown in Fig. 13 show that all boards work correctly, so the frequency can rise to $(250-225)/2=237.5$ MHz. Now, not all the boards behave correctly. Therefore, an additional criterion is needed to decide where to stop in the clock frequency.

As it can be seen in Fig. 13, in the fully reconfigurable architecture, single clock (v3) aging sensors with 7 core AS elements are implemented, and the designer can select which of the OUT_AS_i sensor outputs will be used to stop the frequency adjustment. OUT_AS_1 (the first core element) has the smaller delay element (DE), thus leading to the smallest safety margin. Results are identified as “0” in Fig. 3, showing that for two boards, TMR signal is a more reliable information that the OUT_AS_1 signal; this safety margin is two small for reliable operation. As we move from OUT_AS_1 up to OUT_AS_6, the safety margin is increased, and the final frequency is monotonically reduced. For instance, for the slowest board (due to Process variation), the D383594 board, OUT_AS_6 leads to a 215 MHz clock frequency value. Nevertheless, bear in mind that the clock frequency estimated by simulation is $f_{clk\_sim}=170$ MHz. This means that even with this pessimistic safety margin, a performance improvement of 26.5% is obtained with the monitoring strategy used in this case study.

![Graph](image-url)

**Fig. 13** – Experimental results with five Spartan6 boards, configuring (9 DAQ units, v3 AS) in which the full configurable AFC drives the Clk signal into a frequency defined by the different core AS elements (Fig. 5)

In accordance to the experimental results shown in Fig. 13, Fig. 14 depicts the computed values of the safety margin, associated with the margin until the CUT fails (this is verified by the CRC signature analyzer), and the variable margins obtained when the designer selects one out of the 7 outputs of the programmable sensor. The comparison is done, assuming a minimum safety margin associated with the circumstance of one out of the 3 modules in the TMR architecture fails. Again, if the frequency tuning process is decided by OUT_AS_1, we can see that for the D268948 board and for the D298885 board, a zero or negative phase margin would occur.
VI – CONCLUSIONS

In this report, a methodology for aging effects mitigation in FPGA-based designs is described.

On-chip delay sensors (built with standard FPGA resources) are used for predictive delay testing, and thus for reliable long-term (aging) monitoring. For aging effects mitigation, an automatic frequency controller (AFC) has been developed and used for driving the modulation of the system clock frequency.

Two Local Aging Sensor (AS) architectures are presented, using single and dual clock signals, and their advantages and drawbacks are highlighted. The dual clock AS is a compact structure, but its programmability is hard, and it can have clock domain crossing problems. The single clock AS in its fully configurable architecture has a large overhead, but it can be programmed easily and has no CDC problems. For fix SensorIn, in which the designer selects which total DE is required for a given safety margin, only core AS element \( j \) is retained in circuit synthesis, and thus a much lower overhead is obtained.

The focus of this report is the design and implementation of the automatic frequency controller, demonstrated in a case study with a single clock AS – a CUT with 9 DAQ units, from the PEM system for medical imaging, configured in Virtex5 and Spartan6 boards. Experimental results show that the aging sensors are efficient in predicting safety margin violations. Moreover, aging effects mitigation can be well performed with the proposed AFC module, by correcting the clock frequency to reestablish the time margins, which are essential in safety critical systems.

ACKNOWLEDGEMENT

This work was supported by Portuguese funds through FCT under project PEst-OE/EEI/LA0021/2013 and grant SFRH/BPD/78759/2011.

REFERENCES


### Annex 1 – Spartan6 DCM SP Port Description [23]

<table>
<thead>
<tr>
<th>Port</th>
<th>Direction</th>
<th>Description</th>
<th>DLL</th>
<th>PS</th>
<th>DFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLKIN</td>
<td>Clock Input</td>
<td>Clock input to DCM. Always required. The CLKIN frequency and jitter must fall within the limits specified in the data sheet.</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>CLKFB</td>
<td>Input</td>
<td>Clock feedback input to DCM. The feedback input is required unless the DFS outputs, CLKFX or CLKFX180, are used stand-alone. The source of the CLKFB input must be the CLK0 or CLK2X output from the DCM and the CLK_FEEDBACK must be set to 1X or 2X accordingly. When set to NONE, CLKFB is unused. Ideally, the feedback point includes the delay added by the clock distribution network, either internally or externally.</td>
<td>R</td>
<td>R</td>
<td>Optional</td>
</tr>
<tr>
<td>_RST</td>
<td>Input</td>
<td>Asynchronous reset input. Resets the DCM logic to its post-configuration state. Causes DCM to reacquire and relock to the CLKIN input. Invertible within DCM block. Non-inverted behavior shown below.</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>PSEN</td>
<td>Input</td>
<td>Variable phase-shift enable. Can be inverted within a DCM block. Non-inverted behavior shown below.</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>PSINCDAC</td>
<td>Input</td>
<td>Increment/decrement variable phase shift. Can be inverted within a DCM block. Non-inverted behavior shown below.</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>PSCLK</td>
<td>Clock Input</td>
<td>Clock input to variable phase shifter, clocked on rising edge. When using a global clock buffer, only the upper eight BUFGMUXs can drive PSCLK: BUFGMUX_X2Y1, BUFGMUX_X2Y2, BUFGMUX_X2Y3, BUFGMUX_X2Y4, BUFGMUX_X3Y5, BUFGMUX_X3Y6, BUFGMUX_X3Y7, and BUFGMUX_X3Y8.</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>CLK0</td>
<td>Clock Output</td>
<td>Same frequency as CLKIN, 0° phase shift (i.e., not phase shifted). Always conditioned to a 50% duty cycle on Spartan-6 FPGAs. CLK_FEEDBACK must be set to 1X or 2X to desklaw CLK0. When CLK_FEEDBACK is set to NONE, no phase relationship with CLKIN is present.</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLK90</td>
<td>Clock Output</td>
<td>Same frequency as CLKIN, 90° phase shift (quarter period). Always conditioned to a 50% duty cycle on Spartan-6 FPGAs.</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLK180</td>
<td>Clock Output</td>
<td>Same frequency as CLKIN, 180° phase shift (half period). Always conditioned to a 50% duty cycle on Spartan-6 FPGAs.</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Port</td>
<td>Direction</td>
<td>Description</td>
<td>DLL</td>
<td>PS</td>
<td>DFS</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----</td>
<td>----</td>
<td>-----</td>
</tr>
<tr>
<td>CLK270</td>
<td>Clock</td>
<td>Same frequency as CLKIN, 270° phase shift (three-quarters periods). Always conditioned to a 50% duty cycle on Spartan-6 FPGAs.</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLK2X</td>
<td>Clock</td>
<td>Double-frequency clock output, 0° phase shift. When available, the CLK2X output always has a 50% duty cycle. Either CLK0 or CLK2X is required as a feedback source for DLL functions. Clock Doubler (CLK2X, CLK2X180) output.</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLK2X180</td>
<td>Clock</td>
<td>Double-frequency clock output, 180° phase shift. When available, the CLK2X180 output always has a 50% duty cycle. Clock Doubler (CLK2X, CLK2X180) output.</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLKDV</td>
<td>Clock</td>
<td>Divided clock output, controlled by the CLKDV_DIVIDE attribute. The CLKDV output has a 50% duty cycle unless the CLKDV_DIVIDE attribute is a non-integer value.</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLKFX</td>
<td>Clock</td>
<td>Synthesized clock output, controlled by the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes. Always has a 50% duty cycle. If the CLKFX or CLKFX180 clock outputs are used standalone, then no clock feedback is required.</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLKFX180</td>
<td>Clock</td>
<td>Synthesized clock output CLKFX, 180° phase shift (appears to be an inverted version of CLKFX). Always has a 50% duty cycle. If only CLKFX or CLKFX180 clock outputs are used on the DCM, then no feedback loop is required.</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATUS[0]</td>
<td>Output</td>
<td>Variable phase shift overflow. Control output for variable fine phase shifting. The variable phase shifter has reached a minimum or maximum limit value. 0: The phase shift has not yet reached its limit value 1: The phase shift has reached its limited value</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATUS[1]</td>
<td>Output</td>
<td>CLKIN Input Stopped Indicator. Available only when the CLKFB feedback input is connected. Held in reset until the LOCKED output is asserted. Requires at least one CLKIN cycle to become active. 0: CLKIN never toggles 1: CLKIN input is not toggling even though the LOCKED output can still be High</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>STATUS[2]</td>
<td>Output</td>
<td>CLKFX or CLKFX180 output stopped indicator. Held in reset until the LOCKED output is asserted. 0: CLKFX and CLKFX180 outputs are toggling 1: CLKFX and CLKFX180 outputs are not toggling, even though the LOCKED output can still be High</td>
<td>R</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 2-6: DCM_SP Ports (Cont’d)

<table>
<thead>
<tr>
<th>Port</th>
<th>Direction</th>
<th>Description</th>
<th>DLL</th>
<th>PS</th>
<th>DFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCKED</td>
<td>Output</td>
<td>All DCM features have locked onto the CLKIN frequency. Clock outputs are now valid, assuming CLKIN is within specified limits. 0: DCM is attempting to lock onto CLKIN frequency. DCM clock outputs are not valid 1: DCM is locked onto CLKIN frequency. DCM clock outputs are valid. 1-to-0: DCM lost lock. Reset DCM. The FPGA waits for all DCMs and PLLs to be locked when LCK_CYCLE is set to control the startup cycles without setting the STARTUP_WAIT attribute on any DCM_SP port. In the starting configuration sequence, the global 3-state (GTS) must be deasserted for the DCM to lock.</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>PSDONE</td>
<td>Output</td>
<td>Variable phase shift operation complete. 0: No phase shift operation is active, phase shift operation is in progress, or RST has been asserted. 1: Requested phase shift operation is complete. Next variable phase shift operation can commence.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
1. DCM clock outputs must use either a horizontal clock (default) or a global clock buffer.
### Table 2-8: DCM_CLKGEN Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLKin</td>
<td>Clock Input</td>
<td>Clock input to DCM. Always required. The CLKin frequency and jitter must fall within the limits specified in the data sheet. In the case of free-running oscillator mode, running clock needs to be connected until DCM is locked and DCM is frozen, then clock can be removed. In the other modes, a free running clock needs to be provided and remains.</td>
</tr>
<tr>
<td>RST</td>
<td>Input</td>
<td>Asynchronous reset input. Resets the DCM logic to its post-configuration state. Causes DCM to reacquire and relock to the CLKin input. Invertable within DCM block. Non-inverted behavior shown below. 0: No effect 1: Reset DCM block. Hold RST pulse high for at least three valid CLKin cycles.</td>
</tr>
<tr>
<td>FREEZE_DCM</td>
<td>Input</td>
<td>Prevents tap adjustment drift in the event of a lost CLKin input. The DCM is then configured into a free-run mode.</td>
</tr>
<tr>
<td>CLKFX</td>
<td>Clock Output</td>
<td>Synthesized clock output, controlled by the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes. Can be either statically set or dynamically programmed through a dedicated 4-wire SPI port (PROCDATA, PROGCLK, PROGIDONE, and PROCGEN). Always has a 50% duty cycle. ( F_{CLKFX} = F_{CLKIN} \cdot \frac{CLKFX_MULTIPLY}{CLKFX_DIVIDE} )</td>
</tr>
<tr>
<td>CLKFX180</td>
<td>Clock Output</td>
<td>Synthesized clock output CLKFX, 180° phase shift (appears to be an inverted version of CLKFX). Always has a 50% duty cycle. ( F_{CLKFX180} = F_{CLKFX} )</td>
</tr>
<tr>
<td>LOCKED</td>
<td>Output</td>
<td>Synchronous output indicates whether the DCM is ready for operation. 0: DCM clock outputs are not valid 1: DCM is ready for operation 1.4-0. DCM lost LOCK. Resets DCM. The FPGA waits for all DCMs and PLLs to be locked when LCK_CYCLE is set to control the startup cycles without setting the STARTUP_WAIT attribute on any DCM_SP port. In the starting configuration sequence, GTS must be deasserted for the DCM to lock.</td>
</tr>
<tr>
<td>STATUS[2]</td>
<td>Output</td>
<td>CLKFX or CLKFX180 output stopped indicator. 0: CLKFX and CLKFX180 outputs are not toggling 1: CLKFX and CLKFX180 outputs are not toggling, even though the LOCKED output can still be High.</td>
</tr>
<tr>
<td>STATUS[7:3, 0]</td>
<td>Output</td>
<td>Reserved. Used for simulating reset circuitry in lower-power Spartan-6 devices (-1L, SIMPRIMS only.</td>
</tr>
<tr>
<td>CLKFXDV</td>
<td>Clock Output</td>
<td>Divided CLKFX output clock. Divide value derived from CLKFXDV_DIVIDE attribute. There is no phase alignment between CLKFX and CLKFXDV. ( F_{CLKFXDV} = \frac{F_{CLKFX}}{CLKFXDV_DIVIDE} )</td>
</tr>
</tbody>
</table>
### Table 2-8: DCM_CLKGEN Ports (Cont’d)

<table>
<thead>
<tr>
<th>Port</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
</table>
| PROGDONE | Output   | Indication of M and D programming completion.  
0: No programming operation of M and D is active, the programming is in progress, or RST has been asserted.  
1: Requested programming is complete. Next PROCCLK operation can commence. |
| PROCDATA | Input     | Serial data input to supply information for the programming of M and/or D values of the DCM. During programming, the value shifted in will be M–1 or D–1 and will begin with the LSB. This input must be applied synchronous to the PROCCLK input. |
| PROCEN  | Input     | Indication of whether the programming of M and D values is enabled or not.  
0: Programming of CLKFX_MULTIPLY and CLKFX_DIVIDE is disabled.  
1: Programming of CLKFX_MULTIPLY and CLKFX_DIVIDE is enabled. |
| PROCCLK | Input     | Clock input for the programming of M and D values.  
When using a global clock buffer, only the upper eight BUFCMUXes can drive PROCCLK: BUFCMUX_X2Y1, BUFCMUX_X2Y2, BUFCMUX_X2Y3, BUFCMUX_X2Y4, BUFCMUX_X3Y5, BUFCMUX_X3Y6, BUFCMUX_X3Y7, and BUFCMUX_X3Y8. |

**Notes:**
1. DCM clock outputs must use either a horizontal clock (default) or a global clock buffer.

**Table 2-9** lists the DCM_CLKGEN attributes. All attributes can be set at design time and programmed during configuration. Use `<ATTRIBUTE>`:`<SETTING>` as appropriate in the design entry tool to set an attribute. The CLKFX_MULTIPLY and CLKFX_DIVIDE are allowed to be changed by the FPGA application at run-time.