Designing Multiple PID Controllers Based on an FPGA for Controlling the Temperature of TEM-cell Surfaces

Vincent Dubreuil
School of Engineering ENSEIRB-MATMECA
National Polytechnic Institute, INP
Bordeaux, France
vdubreuil@enseirb-matmeca.fr

Artem V.Osintsev
Department of Data Processing Automation
Tomsk State University of Control Systems and Radioelectronics, TUSUR
Tomsk, Russian Federation
kubenet@gmail.com

***Abstract*—A field programmable gate array (FPGA) has been configured to control the temperature average value and spatial uniformity of a TEM-cell. The system is composed of 24 Peltier elements (PE) and 52 temperature sensors (TS) distributed all around the TEM-cell. The FPGA controls each PE independently computing a proportional-integral-derivative (PID) controller with 24 different sets of parameters. The 52 TS are read using 18 Inter-Integrated-Circuit buses wired to a single master controller. This allows running the PID controller for the 24 sub-systems at a maximum frequency of 2 kHz. To compute this, a custom programmable architecture has been designed to carry out calculations with 36-bit signed values with a fixed point. The final design uses around 1600 logic elements (one logic element = one register and one 4-input lookup table), some multipliers, and some memory bits. The temperature precision obtained is around 0.1°C. The component used is an EP2C5T144C8 configured in a hardware description language Verilog with the Quartus software.**

Keywords—FPGA, PID-controller, PWM, multiphase, I2C, Peltier Element

I. INTRODUCTION

Control systems usually have to control many objects at the same time Data about objects can be received both from a user and various control devices. The results of data processing are displayed or saved in the device memory. However, the processing speed is not as crucial for input/output devices as it is for the algorithm of pulse-width modulation (PWM) signals [1]. Such signals are used to control motors in various devices such as unmanned aerial vehicles [2], robotic systems [3] etc. Control systems of such devices must quickly respond to changes in the parameters of the object and calculate control parameters to maintain its functioning. The calculated result depends both on system parameters and external environment, such as temperature [4, 5], humidity, etc. Regarding temperature, it can be essential to control its spatial distribution in addition to its average value. This can be obtained using several sensors and actuators all around the object to control. The more actuators and sensors we have, the more we increase the spatial precision, but this dramatically complicates the global control system. Moreover, to have a good local temperature precision, it is necessary to use some control algorithms such as a proportional-integral-derivative (PID) controller. This kind of algorithm is widely used because it is efficient and easy to implement on a microcontroller (MC). But in this application, one MC does not have enough functionallity to control such large number of sensors and actuators.

The research was carried out at the expense of Russian Science Foundation grant 19-79-10162 in TUSUR University.

The problem was encountered designing a temperature controller for a TEM-cell to study electromagnetic impact of components in appropriate climatic conditions [6]. Running simulations, it has been found out that 24 Peltier Elements (PE) around the cell surface can regulate its temperature from -50°C to +150°C [6]. Thus, to control each element individually, 24 different PWM signals had to be generated. Additionally, temperature sensors (TS) had to be in contact with the external surface of the TEM-cell, so it was impossible to have one TS for one PE. Thus, it was chosen to place 52 TSs all around the cell, next to the PEs. The temperature of the surface in contact with one PE was then calculated by the average of several sensors. The TS used communicated with the Inter-Integrated-Circuit (I^2C) protocol, which allowed having several units on the same bus. But because of design constraints, only 3 TSs could be wired to one bus, so 18-bus I^2Cs were required to control all the TSs. A common MC cannot handle such number of devices. Therefore, the first solution was to pick several MCs to divide the tasks. Using multiple components has the disadvantage of the necessity to synchronize them all, but there is a solution to the problem. This solution allows having as many PWM outputs, and as many I^2C controllers as we want, but this creates another problem. If two PEs sharing the same TS (as the temperature of a PE is calculated from several TSs) are controlled by two different MCs, the latter have to communicate with each other to share the temperature measurement from the TS. With 52 TSs, this can be very difficult to handle. So, the idea we explored was to use only one component to control all the sensors and actuators. A field programmable gate array (FPGA) seemed to be a good solution to control such large amount of PWMs and I^2Cs as we can use it to define any hardware component we want. With such technology, the implementation of the PID-controller is feasible, although it is less easy [7] because computing this algorithm takes 24 times for the 24 PEs. The main goal of this research was to design multiple PID controllers based on an FPGA to control the temperature of TEM-cell surfaces.

II. GENERATION OF THE PWM SIGNALS

Generating PWM signals with an FPGA is quite basic, but our application had some specific requirements. The goal here was to control PEs with 24 fixed frequency PWM signals, by adjusting duty cycles. However, as the current consumption of a PE can be greater than 10 A at 12 V, powering a large amount of such PEs simultaneously requires to have a power supply with an extremely large yield. To reduce the maximum possible current load, it was chosen to distribute PWM signals over 4 phases to have a maximum of 6 PEs powered at the same time. Thus, this choice also limits the maximum duty cycle at 25%.

Moreover, it was necessary to configure the PWM from 100 Hz to 2 kHz to be able to choose the best conditions for the H-bridges. To generate a PWM signal with an FPGA, the quickest idea is to design a counter dividing the master clock of the FPGA to reach the PWM frequency required, and then to wire a variable comparator to the counter to output a HIGH signal if the counter is between 0 and the comparator value, and LOW in the opposite case. With a master clock at 50 MHz, and a PWM frequency at 100 Hz, this requires implementing a 19-bit counter to have at least a division factor of 500 k. With 24 such components, we then already use at least 24×19×2 = 912 registers for the counters and the comparators, what is consequent. We found a better solution. As all PWM signals have the same frequency, the idea was to use the same counter for the 24 signals, but 24 different comparators, which is more efficient. Additionally, as the signals have to be on 4 phases that do not overlap, comparators can be only 17-bits long, comparing with 17 less significant bits of the counter, and being deactivated for 3 phases out of 4. Then, this system uses only 24×17+19 = 427 registers (excluding registers for general control), which is more than twice efficient than the previous solution. As the implementation is parametric, the system can operate with different frequencies, number of PWM signals, and number of phases for other applications.

III. Reading the Temperature Sensors

The TS used communicates with the I²C protocol. In our application, the address of the TS can be adjusted by hardware, wiring some pins of the component to the power supply or the ground and allowing up to 32 different addresses. Thus, in theory, 2 buses would be enough to control the 52 TSs. However, the maximum I²C communication speed of the TS is 400 kHz, so it takes 125 µs to read one measurement. As we wanted a maximum operating frequency of 2 kHz, only 4 TSs in a row could be read in one period. Using buses with 3 TSs in each, the timing requirement can be fulfilled. We also used 3 TSs per I²C as it simplifies their wiring: printed circuit boards (PCB) were designed to receive 3 TSs, by setting them with 3 different addresses. The same PCB could be used to wire all the TSs. Again, the first idea here was to design one I²C controller and to use it 18 different times. But as the PWM generator, we can group the 18 buses for one controller. In each bus, we have 3 TSs with 3 different addresses, but between the 18 buses, these 3 addresses are the same. Thus, the requests to read the 3 temperature values on one bus are the same requests for all the other buses. Therefore, we can have one controller sending the same requests for the 18 buses but getting temperature values independently. Fig. 1 represents this structure.

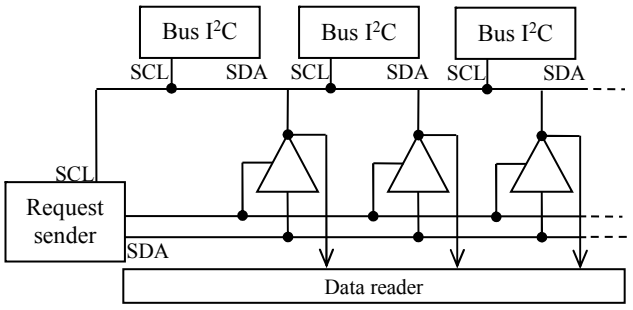

Fig. 1. Structure of the I²C controller

Using the 3-state buffers of the output pins of the FPGA, we can send the same requests to the 18 buses and collect data on the 18 different data lines. Thus, this system allows reading temperature values from 18 buses in a very efficient way.

IV. Calculation of the PID Algorithm

A. Mathematical model

First, we had to find exactly what calculations needed to be done. We modelled the control system of one PE as в виде) a simple loop with the feedback as described in Fig. 2.

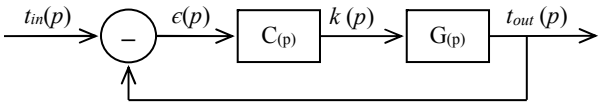

Fig. 2. Block diagram of one Peltier element control system

$G(p)$ represents the H-bridge, the PE and the TEM-cell, and $C(p)$ is the regulator we had to design. t_{in} is the temperature command that the system has to reach, t_{out} is the actual temperature measured locally around the PE, ε is the error, the difference between t_{in} and t_{out}, the system has to regulate this value around 0, k is the duty cycle command calculated by the corrector, to be fed to the input of the H-bridge. In the continuous frequency range, the corrector $C(p)$ can be defined by the expression below (1) [8] implementing the 3 operations of the PID controller: k is the proportional coefficient, ω_i is the pulsation limit of the integral effect, ω_1 and ω_2 are the pulsation limits of the derivative effect.

$$C(p) = K \frac{1 + \frac{p}{w_i}}{\frac{p}{w_i}} \times \frac{1 + \frac{p}{w_1}}{1 + \frac{p}{w_2}} \quad (1)$$

To continue, the expression (1) has to be converted in the discrete time range. But first we need to define sample frequency fs, at which the PID controller works. In our application, it is typically between 100 Hz and 2 kHz. In the discrete-time range we then consider the variables t_{in}, t_{out}, ε and k as values sampled at f_s. Thus, we use the notations $t_{in}(n)$, $t_{out}(n)$, $\varepsilon(n)$ and $k(n)$ for the n-th samples of the corresponding continuous-time variables. Thus, using for example the Tustin's bilinear transform, we can obtain (2) (with the Z-transform), where α, β, γ, a and b are parameters directly defined by K, ω_i, ω_1 and ω_2. With the inverse Z-transform, we finally get (3):

$$C(z) = \frac{\alpha z^{-2} + \beta z^{-1} + \gamma}{a z^{-2} + b z^{-1} + 1} \quad (2)$$

Equation (3) shows that the new samples of the command k can be calculated from the previous samples of command k and the samples of the error ε with 5 multiplications and 4 additions. This is the calculation we had to implement.

$$k(n) = \alpha \in (n-2) + \beta \in (n-1) + \gamma \in (n) - ak(n-2) - bk(n-1) \quad (3)$$

B. The 36-bits core

To calculate (3), we designed a custom programmable architecture. We first designed a logic core to compute mathematical operations. The FPGA we used had some [18×18] bits multipliers, and as 18 bits were not enough to

have a sufficient precision, we chose to represent data on 36 signed bits with a fixed point. Using 4 [18×18] multipliers and some logic elements (registers and lookup tables), we designed a multiplier-accumulator [36×36] bits to 72 bits. This element is the arithmetic logic unit (ALU) of the architecture as represented in Fig. 3. Thus, it allows calculating the sum of multiplications, exactly what we needed to calculate (3).

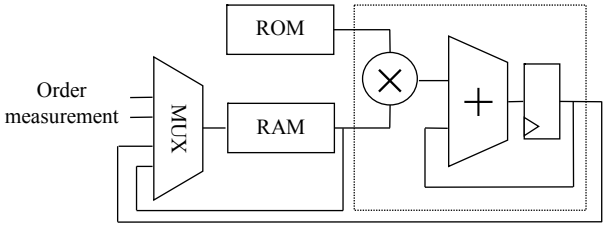

Fig. 3. 36-bit core with an ALU and two data memories

On the first input of the ALU, we put a read only memory (ROM) block containing 128 words of 36 bits. This memory stores the parameters α, β, γ, a and b (from (3)) for each PE (5×24 = 120 values) and 8 special values at the end (explained later). On the second input of the ALU, we put a random access memory (RAM) block containing also 128 words of 36 bits. This memory stores the variables $\varepsilon_{(n-2)}$, $\varepsilon_{(n-1)}$, $\varepsilon_{(n)}$, $k_{(n-2)}$, $k_{(n-1)}$ for each PE. The input of this RAM block is controlled by a 4 channel multiplexer (MUX). The first channel comes from the output of the RAM: wisely controlling addresses, this allows copying values inside the RAM. The second channel comes from the output of the ALU: this allows storing results of calculations back to the RAM. The third channel is the input of temperature measurements from the I^2C controller (bus). The fourth channel is the input of the desired temperature t_{in}. In the 8 additional words available at the end of the ROM, we put some special values such as "1" and "-1" to be able to load values from the RAM to the accumulator of the ALU. This allows bypassing the multiplier effect to use only the accumulator. This way, we can also make simple additions (taking "1" from the ROM) and subtractions (taking "-1" from the ROM). Controlling bus addresses and clock enables of these different modules, we can compute one step of the PID controller with a few sub-steps. First, we have to calculate the error $\varepsilon_{(n)}$ from the difference of the temperature measurement obtained on the third input of the MUX and the temperature requested on the fourth input of the MUX. This is done by transiting the temperature requested through an unused RAM word to the ALU, multiplying it by "1" to set the accumulator value to this temperature value. Then, we do the same thing for the temperature measurement, but multiplying it by "-1", so we finally subtract it to the temperature value requested. At this point, the accumulator contains the actual temperature error $\varepsilon_{(n)}$. This value is then written back to the RAM to its appropriate location. After that, we can start the calculation of (3). This is computed by reading the parameters α, β, γ, a, b, from the ROM and the variables $\varepsilon_{(n-2)}$, $\varepsilon_{(n-1)}$, $\varepsilon_{(n)}$, $k_{(n-2)}$, $k_{(n-1)}$, from the RAM in a row to add 5 multiplication results from these numbers, according to (3). At the end of this operation, the result is then stored in the accumulator. But before writing it back to the RAM, we have to move some values inside this memory. Using the first input of the MUX, we copy $\varepsilon_{(n-1)}$ to $\varepsilon_{(n-2)}$, $\varepsilon_{(n)}$ to $\varepsilon_{(n-1)}$, and $k_{(n-1)}$ to $k_{(n-2)}$, to prepare the next calculation step of the PID controller algorithm. Then we can store the result in the accumulator to the RAM to $k_{(n)}$. At this moment, we also send this new command $k_{(n)}$ to the PWM generator (which is wired to the output of the RAM) to update the corresponding PWM signal. This process is then executed for the 24 sub-systems to independently control each PE.

C. Temperature measurement management

As explained in the introduction, we have 52 TSs distributed around the PEs. Hence, to know the temperature of one PE we have to calculate the average value of several temperature measurements. However, this calculation is also carried out by the 36-bit core, that is why no big architecture modifications were needed. Temperature measurements from the I^2C controller (bus) are in fact stored in another data memory called "TEMP_MEM". The I^2C controller stores these values in a row following the order of reading, without spatial correlation. Then this memory read port is wired to the third input of the MUX. Therefore, wisely controlling the read address of the TEMP_MEM, we can let the core access any temperature measurements. Thus, at the beginning of a new step in the PID controller algorithm, even before the calculation of the error ε, we first have to calculate the average of several temperature measurements. In reality, 16 PEs are surrounded by 4 TSs and 8 PEs by only 2 TSs. To simplify the problem, we considered that the PE with 2 TSs had virtually 4 TSs, duplicating the value measured in the TEMP_MEM. Thus, we calculate the average of 4 measurements. The sum is directly obtained thanks to the ALU. Then, when this average is computed, the process described in the preceding sub-section can be executed. Including this average calculation, computing one PID algorithm takes exactly 24 sub-steps. As the FPGA used runs at 50 MHz, one PID controller algorithm is computed in exactly 480 ns. So the 24 PID controller algorithms are computed in around 12 µs.

D. Control of the architecture

Until now, all the processes were described considering that we can "wisely" control addresses of memories and different control signals, but obviously we also had to design this architecture controller. It was chosen to make it programmable. This makes possible to quickly change the behaviour of the system to perform specific tests on a particular part of the architecture. For example, Fig. 4 represents it. First, in the centre, we have a memory block called "PROG_MEM" that stores our function; in fact, several functions can be stored at the same time (Fig. 4).

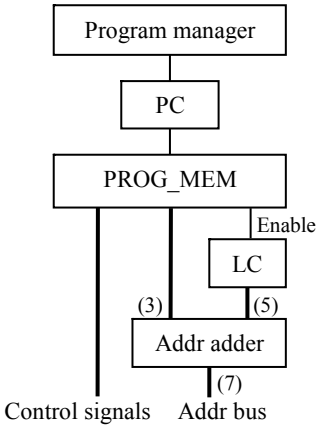

Fig. 4. Controller architecture

Each word of the memory represents an instruction. An instruction is directly composed of control signals controlling

the architecture, except for the address bus. The address bus is a 7 bit wide bus controlling the address data read and write operations (7 bits are enough to address the 128 words of the ROM/RAM). It is defined by 3 signals coming from the PROG_MEM, and 5 signals coming from a 5 bit counter named "loop counter" (LC) enabled by another signal from the PROG_MEM. If we name the 3 bit value v, and the 5 bit value w, then the address adder combines these two values to produce the address bus value which is equal to $v+5\times w$. This allows having a program that computes the PID controller algorithm for one sub-system, that we loop 24 times, each time incrementing the LC to execute the program on different locations of the address data. Thus, we have a single program working for 24 sub-systems. The PROG_MEM is controlled by another counter named program counter (PC). When this counter increments, the next instruction from the memory is read. The PC is controlled by a last module named program manager. This module can set the initial value of the PC to different values to start the program at different locations in the PROG_MEM. This allows storing different programs and executing them whenever we want, just controlling the program manager.

E. External communication and task management

To debug the system, and to observe its evolution while working, we wanted to send the 52 temperature measurements and the 24 duty cycles commands out of the FPGA at each PID controller iteration. To do so, we designed a module implementing the universal asynchronous receiver transmitter (UART) protocol with an operating frequency of 10 MHz. This allows sending bytes at 1 MHz. We wired the data input of this module to the data output of the RAM. Thus, always using the same architecture, we can send whatever data we want. The problem is that we cannot use this module at the same time as we compute PID controller iterations, otherwise it would dramatically decrease the computation speed, as sending just one duty cycle value takes 5 µs. Therefore, we developed different programs executed at different moments. Fig. 5 represents different tasks executed in one PWM period in time domain. Fig. 6 is a flowchart of these tasks. First, the I²C controller is launched to read the 52 TSs. When this task is done, the PID controller program is launched. Then the system waits for the PWM cycle to end to start another cycle. However, we also used the moment while sensors are read to execute two other programs to send data via UART. The first program sends the 52 temperature measurements from the previous cycle. The program sends these measurements in the same order that the I²C controller wrote them.

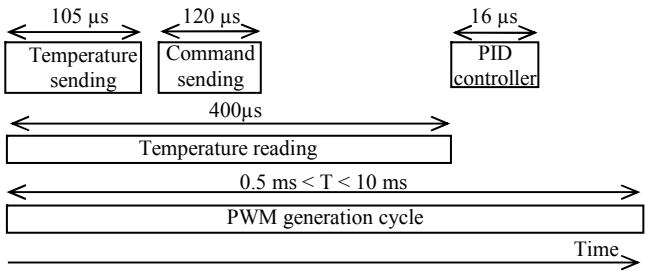

Fig. 5. Time diagram of different tasks executed in one PWM period

As the program runs faster than the I²C controller, it has enough time to send measurements before they are overwritten. After that, another program sends the 24 previous duty cycles commands by reading them in the RAM (Fig. 5, 6).

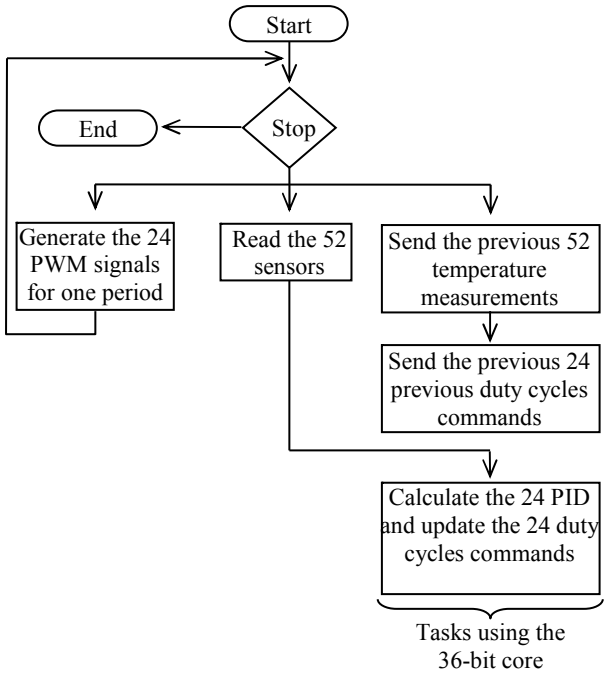

Fig. 6. Flowchart of different tasks executed in one PWM period

To achieve this functionality, we had to respect some timing constraints. First, reading sensors and calculating the PID controller had to be finished before the end of the PWM cycle, otherwise, this would have extended the period of the PWM, which is not desirable. In addition, the two sending data programs had to be finished before the end of the sensor reading task, otherwise the calculation of the PID controller program (functional block) could not have been started in time. These constraints mainly fixed the operating frequency of the UART transmitter. Moreover, another module implementing the UART protocol was added, but to get commands from the outside of the FPGA (Receiver). This module allows starting and stopping the system, and setting the temperature order that the system has to reach.

V. RESULTS

The system was completely described in a hardware description language Verilog, and implemented on an FPGA EP2C5T144C8 with a master frequency of 50 MHz. The developed system employed 1600 logic elements (35%), including around 1000 registers and 1350 4-input lookup tables (LUT4), 16 kbits of memory (14%), and 4 embedded multipliers [18×18] (from 13 available). The specific resource use of each part of the system is described in Table I bellow.

TABLE I. RESOURCE USE OF DIFFERENT SYSTEM PARTS

Module	LUT4	Registers
ALU	172	55
MUX	93	0
PWM generator	501	453
I2C controller	316	309
UART rx/tx	152	165
Other control	99	27
Total	1333	1009

The biggest system parts are the PWM generator and the I²C controller, which used 60% of the LUT and 75% of all used registers. This is because the architecture uses mainly

memory and multipliers which are designed in the FPGA itself, but does not use any basic logic elements. The system was tested with PEs and TSs. We got some PID parameters allowing temperature control with a precision of 0.1°C, which is sufficient for the application. With an oscilloscope, we experimentally observed different signals out of the FPGA to confirm our timing calculations (Fig. 7).

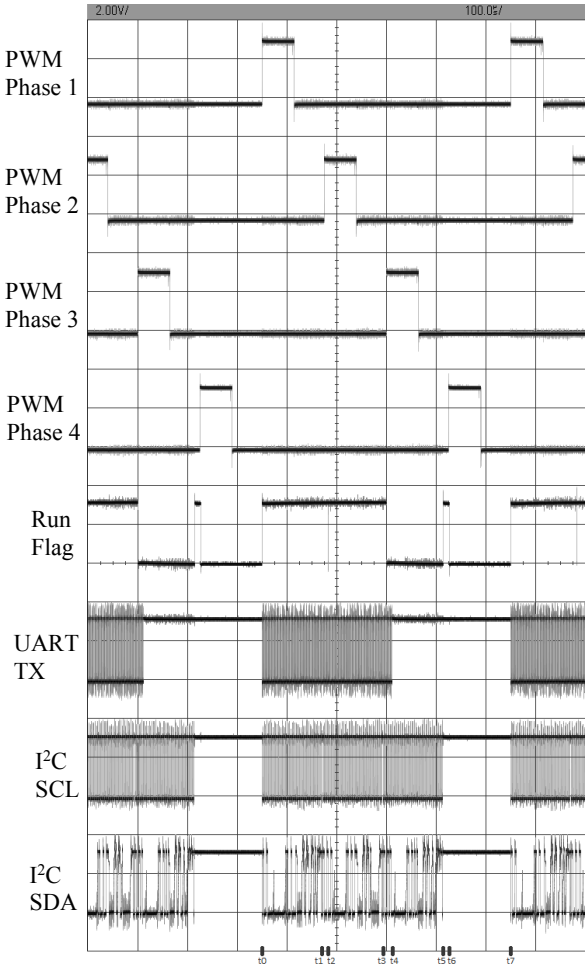

Fig. 7. Signals observed with an oscilloscope while system working

Measurements were taken at the maximum frequency of the developed system, namely 2 kHz. A complete cycle starts at t_0 and ends at t_7 (t_7–t_0 = 500 µs). The first 4 signals are four PWM signals from each different phase. The "program run flag" signal indicates when the program is running. At t_0 the first program starts to send temperature values. At t_2, this first program ends and the second one starts immediately (thin spike downward). At t_4, the second program ends. This activity is also visible looking at the UART TX signal used to send data values: the UART is active just while these two programs are runing. At the same time, TSs are also read. These reading operations are visible looking at the I²C SCL and SDA signals. This task starts at t_0 and ends at t_5. We can also observe the transitions between the 3 requests (because of the 3 sensors per bus) at t_1 and t_3 on the I²C SCL signal (little white gap). As soon as this task is finished at t_5, the third program computing 24 PID controllers starts ("program run flag" signal) and ends at t_6. At this point, all calculations are made and the system has just to wait for the PWM generation cycle to end to begin a new cycle. The time t_7–t_6 ≈ 100 µs is sort of "free time". Then, in theory, the system could work with an operating frequency around 2.5 kHz, just by reducing this "free time". With lower frequencies, the system also works in the same way, just increasing the length of this "free time". We can clearly see that the limiting factors are the external communications such as the I²C and the UART. The calculation of the PID controller takes only 2.5% of the operating period at 2 kHz, against 75% for the I²C and 45% for the UART. These measurements confirm that the system is well sized to operate at 2 kHz, as all the timings are in agreement with the theoretical timings calculated and represented in Fig. 5. The task scheduling constraints are also respected, as the 2 programs sending data end before all the sensors are read, and the calculation of the PID controller ends before the end of the PWM generation cycle.

VI. Conclusion

We chose an FPGA to control 24 PEs and 52 TSs and designed a module able to generate PWM signals with the same frequency but with different duty cycles and phases. This enables individual control of each PEs divided in 4 phases to reduce the maximum current peak consumption. In addition, we designed a module controlling 18 different I²Cs to get the temperature from 52 TSs in less than 400 µs and designed a programmable architecture to make additions and multiplications with 36 bit values. This allows computing one PID controller in about 500 ns. The algorithm is executed for each PE with different parameters taking around 12 µs to execute. This allows adjusting parameters of the PID controller for each PE to get the best spatial temperature control. The system can control the temperature with a precision of 0.1 °C. It was implemented on an EP2C5T144C8. This system was developed for temperature control, but it can be adapted for other control systems involving multiple PID controller calculations.

References

[1] M.V. Balagurov, A.V. Sidorov, D.V. Korobkov, M.A. Zharkov, D.A. Shtein, I.O. Bessonov, "The universal control system for semiconductor converters with PWM". 16-th Int. Conf. of Young Specialists on Micro/Nanotechnologies and Electron Devices EDM 2015: Conference Proceedings, 2015, pp. 379–383.

[2] N. Monterrosa, "Design and implementation of a motor control module based on PWM and FPGA for the development of a UAV flight controller". CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies. (CHILECON) pp. 783–789.

[3] M. Ho, J.P. Desai, "Towards a MRI-compatible meso-scale SMA-actuated robot using PWM control". Conference on Biomedical Robotics and Biomechatronics (BioRob). 3rd IEEE RAS and EMBS International, 2010, pp. 361–366.

[4] Z. Enjing, Y. Huijuan, F. Jian, J. Xue, H. Dandan, "The Design of Temperature Control System of Test Cell Based on Predictive Control Algorithm", 2014 Fourth International Conference on Instrumentation and Measurement, Computer, Communication and Control, Sept. 2014, pp. 226–229.

[5] A. I. Lita, D. A. Visan, L. M. Ionescu, A. G. Mazare, "Temperature Control System for Accelerated Aging Tests on Printed Circuit Boards", 2018 10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), June 2018, pp. 1–4.

[6] A. Osintsev, A. Sobko, M. Komnatnov, "Temperature controller for external surface of waveguide", 2016 International Siberian Conference on Control and Communications (SIBCON), May 2016, pp. 1–4.

[7] K. Lee, Y. Kim, "Design and Analysis of Digital PID Controller in MCU and FPGA", 2018 International SoC Design Conference (ISOCC), Nov. 2018, pp. 261–262.

[8] I. El-sharif, F. Hareb, A. Zerek, "Design of discrete-time PID controller", International Conference on Control, Engineering & Information Technology (CEIT'14), 2014, pp. 110–115.