IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 26 September 2022, accepted 30 October 2022, date of publication 7 November 2022, date of current version 15 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3220660

APPLIED RESEARCH

Design and Implementation of Asynchronous Processor on FPGA

ZIHO SHIN[1] **AND MYEONG-HOON OH**[2]
[1]The Affiliated Institute of Electronics and Telecommunication Research Institute (ETRI), Daejeon 34129, Republic of Korea
[2]Department of Computer Engineering, Honam University, Gwangju 62399, Republic of Korea

Corresponding author: Myeong-Hoon Oh (mhoh@honam.ac.kr)

This study was supported by research fund from Honam University, 2021.

ABSTRACT Low-power and fault-tolerant features for microprocessors have been recognized as two of the greatest concerns in applications for edge devices. In this study, we propose asynchronous circuit design techniques for field-programmable gate arrays (FPGA) that can fundamentally overcome the drawbacks of conventional synchronous circuit designs. We used commercial FPGAs and implemented an asynchronous MSP430 microprocessor using the proposed technique. We also introduce an interfacing architecture between the synchronous block memory and asynchronous core to support the congeniality of the commercial embedded processor and the asynchronous core. Furthermore, we analyze a compiler for MSP430 and adapt its result to achieve a high-level development environment for asynchronous MSP430. The experimental results showed that the asynchronous MSP430 consumes 62.3% less power than its synchronous counterpart and has significant fault tolerance compared to the synchronous MSP430 under unstable supply voltage conditions. Additionally, the asynchronous MSP430 emitted 13% less electromagnetic noise at the working frequency.

INDEX TERMS Asynchronous circuits, field-programmable gate arrays, microprocessors.

I. INTRODUCTION

Most edge devices typically experience severe external conditions, such as battery-based unstable supply voltage and variations in temperature and humidity. Therefore, applications, for edge devices, require a low-power and fault-tolerant processor capable of reliably running applications under unstable circumstances.

Conventionally, processors for edge devices are largely implemented using synchronous circuit design methodology. Nevertheless, this design methodology has some drawbacks that contradict the requirements for the applications of edge devices, such as excessive power consumption caused by the clock network, performance degradation from clock skew, and meta-stability issues owing to the existence of multiple clock domains [1], [2].

To provide fault-tolerant and low-power characteristics, which are the key features for the processors of edge devices, the design paradigm needs to shift to an alternative methodology that does not use a clock network, namely an asynchronous circuit design methodology.

Major industrial vendors have adopted asynchronous circuit design methodologies to overcome the obstacles of the synchronous circuit design. For the FM6000 series [3], which was introduced by Intel as a high-performance ethernet switch chip, over 90% of the whole system was implemented by employing asynchronous circuit design techniques. The brain-inspired chip: TrueNorth [4], which is designed by IBM as a neuromorphic processor, uses a 'spike' architecture for internal networking. The fundamental idea of the spike system is a handshake protocol from the asynchronous circuit design. The picoPIPE architecture [5] of the Speedster FPGA, which is known as a high-performance FPGA, manufactured by Archronix [6], also accommodates an asynchronous circuit design methodology.

However, implementing an asynchronous circuit is different from implementing a synchronous circuit design. The control logic of an asynchronous circuit is normally

The associate editor coordinating the review of this manuscript and approving it for publication was Paolo Crippa.

implemented using specific graph-based theories, such as a signal transition graph (STG) [7] and an asynchronous finite state machine (AFSM) [8]. Furthermore, to design an asynchronous processor for an edge device as an application-specific integrated circuit (ASIC), a field-programmable gate array (FPGA)-based design methodology is required for the functional verification of the desired platform.

Nevertheless, most commercial FPGA vendors only support generic finite-state machine (FSM)-based designs, which are controlled by a global clock, and they do not support asynchronous circuit designs in their design flow. This is the reason why design techniques for asynchronous circuits on FPGA are in high demand.

In this study, we propose asynchronous circuit design techniques for commercialized FPGA. Using the proposed design techniques, we designed and implemented a microprocessor based on the MSP430 instruction set architecture (ISA) [9].

II. RELATED WORK

Unlike a conventional synchronous circuit, an asynchronous circuit uses a locally distributed control signal, known as handshake protocol, rather than a global signal to control the entire system. The main issues in the design methodology of asynchronous circuits are the handshake protocol and delay model for circuit modeling.

The handshake protocol uses *Request* (*Req*) and *Acknowledgment* (*Ack*) signals, which represent the validity and arrival of data, respectively. According to the triggering method, the handshake protocol can be classified as 4-phase signaling (level signaling, return-to-zero) and 2-phase signaling (transition signaling, non-return-to-zero) [10].

Regarding the delay model, which is another main issue in the design methodology of asynchronous circuits, three delay models have been studied: bounded delay, delay-insensitive, and speed-independent. These delay models have distinctive modeling strategies for the wires and gates of the circuits.

The bounded delay model assumes the limitation of delayed time in the gates and wires. The design methodology based on this delay model is beneficial in terms of implementation because it resembles a synchronous circuit design, in which gates and wires are bounded in a clock cycle. Therefore, in this model, except for the controller, the data path can be used in the same manner as that of the synchronous design technique without modification.

An AFSM controller that can support the extended burst mode (XBM) [11] was implemented on the FPGA with the bounded delay model [12]. However, the authors only observed the feasibility of FPGA implementation for asynchronous control logic, rather than implementing a large-scale system such as a processor.

Similarly, another group implemented the XBM in an FPGA [13]. The authors designed a synchronous circuit with the characteristics of an asynchronous circuit by adapting a locally clocked XBM which utilizes a clock signal to drive the XBM. However, because the clock signal was used as the driving signal for the XBM controller, they did not implement an asynchronous circuit on the FPGA fundamentally.

In the case of the delay-insensitive model, the delays of the wires and gates are modeled as unbounded, which resonates with the ideal characteristics of an asynchronous circuit. Delay-insensitive characteristics can be achieved by encoding a *Req* signal into the data line. Accordingly, the data path needs to utilize a dual-rail or quad-rail encoding scheme rather than the conventional single-rail scheme.

As one of the design schemes based on the delay-insensitive model, null convention logic (NCL) cell libraries were developed by adapting look-up table (LUT)-based programming to support implementation on the FPGA [14]. Because the delay-insensitive system requires a multi-rail encoding scheme, area and complexity issues can occur in the FPGA implementation applied to these cell libraries.

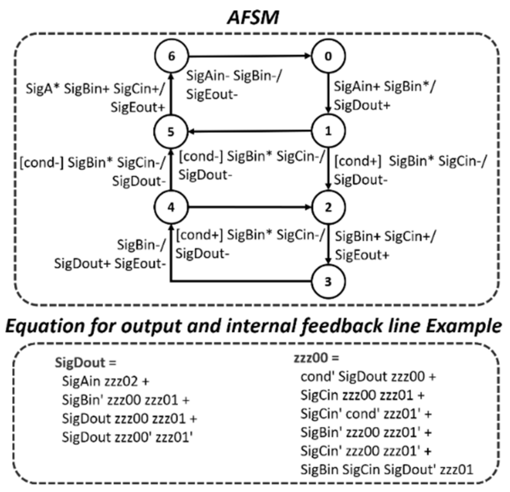
Another research group [15] subsequently solved the area issue, which was a drawback of the previous study [14], by combining the bounded delay model-based single-rail scheme and the dual-rail-based delay-insensitive model. Nonetheless, they did not design delay-insensitive cell libraries, nor did they implement controllers for the bounded delay model. Moreover, as they did not use the proposed design technique to implement a large-scale system, the coverage of the proposed design methodology was opaque [15].

For the speed-independent model, the delay in the wire is assumed to be zero, and the delay of the gate is modeled as unbounded. Thus, the speed-independent model can be more asynchronous than the bounded-delay model. The implementation of an asynchronous microprocessor on an FPGA based on the speed-independent model was introduced in [16] by accommodating the STG-based controller. However, the disadvantage of this approach is that the design of the STG controller is more complex than that of the others because the STG controller requires calculation and the adjustment of timing to detect the transition of control signals. Additionally, the speed-independent model requires arbitration logic (MUTEX) which may adversely affect the design.

In this study, we propose techniques for designing asynchronous circuits on commercial FPGAs. Specifically, an AFSM-based bounded delay model is adapted for implementing the control logic to avoid area issues and exploit the ease of design. We also propose a specialized LUT-style coding scheme that ensures stability in FPGA-based implementation of the control logic. In addition, to efficiently utilize the implemented core, we propose an interfacing logic between block memory, which is embedded in the FPGA, and an asynchronous core.

To verify the coverage of the proposed design techniques, we used the proposed techniques to design and implement an MSP430 processor core which is frequently used in the end node on a commercial FPGA.

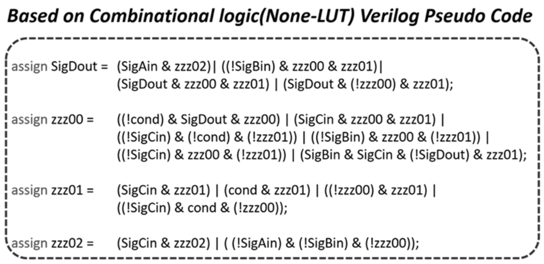
Furthermore, to provide compatibility with a commercial MSP430 microprocessor, we analyzed the compiler and incorporated its results into the designed core.


AFSM
6
0
1
5
4
2
3
SigAin- SigBin-/ SigEout-
SigA* SigBin+ SigCin+/ SigEout+
SigAin+ SigBin*/ SigDout+
[cond-] SigBin* SigCin-/ SigDout-
[cond-] SigBin* SigCin-/ SigDout-
[cond+] SigBin* SigCin-/ SigDout-
SigBin-/ SigDout+ SigEout-
[cond+] SigBin* SigCin-/ SigDout-
SigBin+ SigCin+/ SigEout+
Equation for output and internal feedback line Example

```
SigDout =
  SigAin zzz02 +
  SigBin' zzz00 zzz01 +
  SigDout zzz00 zzz01 +
  SigDout zzz00' zzz01'

zzz00 =
  cond' SigDout zzz00 +
  SigCin zzz00 zzz01 +
  SigCin' cond' zzz01' +
  SigBin' zzz00 zzz01' +
  SigCin' zzz00 zzz01' +
  SigBin SigCin SigDout' zzz01
```

FIGURE 1. Example of the AFSM and boolean equation.


Based on Combinational logic(None-LUT) Verilog Pseudo Code

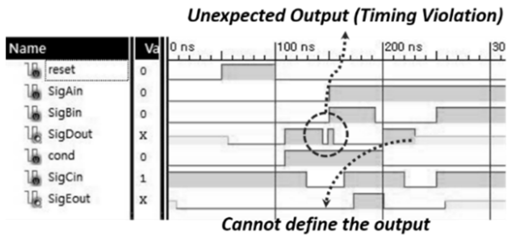
```
assign SigDout =  (SigAin & zzz02)| ((!SigBin) & zzz00 & zzz01)|
                  (SigDout & zzz00 & zzz01) | (SigDout & (!zzz00) & zzz01);

assign zzz00 =    ((!cond) & SigDout & zzz00) | (SigCin & zzz00 & zzz01) |
                  ((!SigCin) & (!cond) & (!zzz01)) | ((!SigBin) & zzz00 & (!zzz01)) |
                  ((!SigCin) & zzz00 & (!zzz01)) | (SigBin & SigCin & (!SigDout) & zzz01);

assign zzz01 =    (SigCin & zzz01) | (cond & zzz01) | ((!zzz00) & zzz01) |
                  ((!SigCin) & cond & (!zzz00));

assign zzz02 =    (SigCin & zzz02) | ( (!SigAin) & (!SigBin) & (!zzz00));
```

(a) Based on combinational logic (Non-LUT) implementation of the AFSM controller



(b) Waveform capture (Post-route): non-LUT-based implementation

FIGURE 2. Example of combinational Logic-based implementation.

III. PROPOSED DESIGN TECHNIQUES FOR AFSM CONTROLLER IN FPGA

In this section, we address the design techniques for the asynchronous controller on a commercial FPGA based on the AFSM by adapting an LUT-based programming style. The interfacing module is also described by identifying the needs of the interfacing module between the asynchronous core and block memory. Finally, a hard macro-based implementation technique is proposed to guarantee the functionality of the implemented asynchronous system.

A. LUT-BASED DESIGN FOR THE ASYNCHRONOUS CONTROLLER

The controller for the asynchronous circuit has multiple feedback lines. In this case, unexpected glitch signals could be generated because of the RACE conditions [17] for each signal. Consequently, this signal would introduce a hazardous condition in the asynchronous circuit. This has been recognized as a critical issue from the perspective of implementing asynchronous controllers.

To explain the detail of the feedback signals in the AFSM-based asynchronous controller, we designed the simple AFSM controller, which is depicted in Fig. 1.

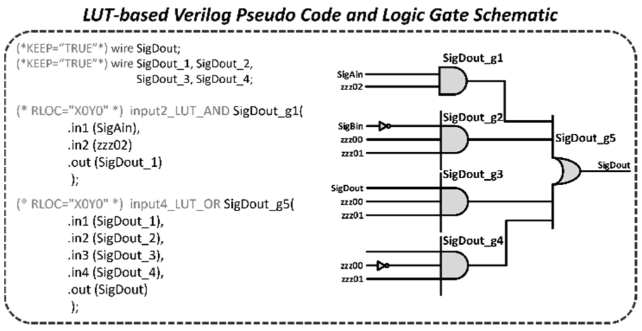
Fig. 1 shows the example of an AFSM for an asynchronous controller. The AFSM is synthesized into gate-level Boolean equations for each output signal using a 3D tool [11]. The results of the output signal (*SigDout*) and internal feedback signal (*zzz00*) are shown in Fig. 1.

Fig. 2 (a) shows an example of generic Verilog pseudo codes for the gate-level Boolean equation from the AFSM in Fig. 1. Such a coding style is commonly used in the design of synchronous-control circuits. However, for an asynchronous controller, implementation based on this coding style could not guarantee a "HAZARD-FREE" characteristic because an unexpected timing violation would occur, as shown in Fig. 2 (b). The reason for this timing violation is that the routing of wires and the placement of logic gates are distributed flexibly by the FPGA design tool, and a logical expression for the given equation specified by the designer can also be optimized by the tool. Therefore, the timing of the feedback loops in each signal can be affected at any specified time.

Unlike the design method above, the LUT-based design is suitable for implementing an asynchronous controller for two main reasons. Firstly, the given equation from the synthesizer basically does not generate "HAZARD" conditions. Secondly, due to the nature of the operating principle of the LUT, the delay of the wire cannot induce the "RACE" conditions. The reason is that the logical expression is set into the LUT directly, and instantiated LUT cells are placed uniquely and relatively [18].

The logical equation for *SigDout* in Fig. 1 can be implemented as a two-level logic, depicted pseudo-code in Fig. 3 (a), with the LUT-based logic gates that the designers should generate, as shown in Fig. 3 (b). To keep and lock the location of the wire and gate, "RLOC (Relative Location)" constraints with the coordinate information of the x-axis and y-axis, such as X0Y0, and "KEEP" constraints are added to the LUT-based code as shown in the left part of Fig. 3 (a). By adopting LUT-based coding, the simulation results depicted in Fig. 3 (c) show that the AFSM controller implemented using the LUT operates well according to the control flow, as shown in Fig. 1.

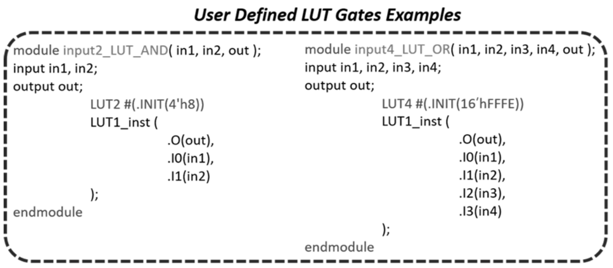
An asynchronous circuit requires specific logic cells such as c-elements [10] to handle the control signals. With regard to the implementation, there are two types of c-elements: the latch-free type which is implemented with logic gates and the latch-style type which is configured with an RS-latch. Nonetheless, neither version of c-elements is supported in the form of libraries using commercial FPGA design tools.



```
(*KEEP="TRUE"*) wire SigDout;
(*KEEP="TRUE"*) wire SigDout_1, SigDout_2,
                     SigDout_3, SigDout_4;

(* RLOC="X0Y0" *)  input2_LUT_AND SigDout_g1(
            .in1 (SigAin),
            .in2 (zzz02)
            .out (SigDout_1)
            );
(* RLOC="X0Y0" *)  input4_LUT_OR SigDout_g5(
            .in1 (SigDout_1),
            .in2 (SigDout_2),
            .in3 (SigDout_3),
            .in4 (SigDout_4),
            .out (SigDout)
            );
```

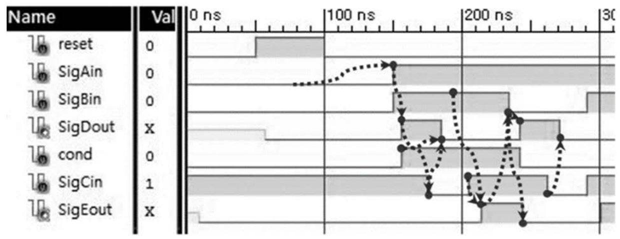
(a) LUT-based Verilog pseudo code for the AFSM implementation


User Defined LUT Gates Examples

```
module input2_LUT_AND( in1, in2, out );
input in1, in2;
output out;
          LUT2 #(.INIT(4'h8))
          LUT1_inst (
                    .O(out),
                    .I0(in1),
                    .I1(in2)
          );
endmodule
```

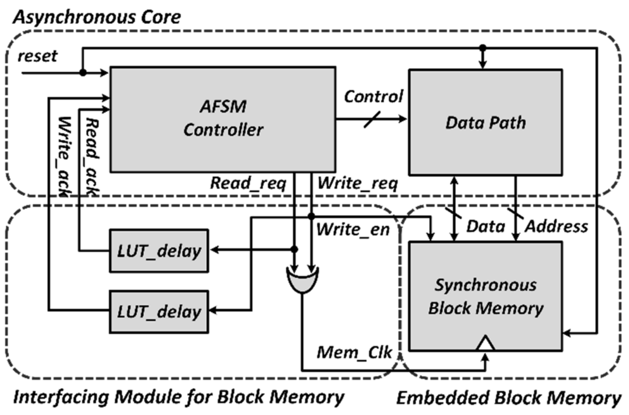
```
module input4_LUT_OR( in1, in2, in3, in4, out );
input in1, in2, in3, in4;
output out;
          LUT4 #(.INIT(16'hFFFE))
          LUT1_inst (
                    .O(out),
                    .I0(in1),
                    .I1(in2),
                    .I2(in3),
                    .I3(in4)
          );
endmodule
```

(b) User-defined LUT gates

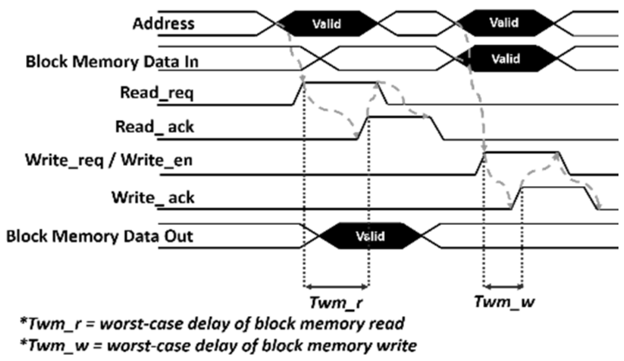


(c) Waveform capture (Post-route): LUT-based implementation

FIGURE 3. Example of LUT-based implementation.



(a) Architecture of the block memory interface



(b) Timing diagram for the block memory interface

FIGURE 4. Architecture for block memory interfacing and timing diagram.

In this study, we provide both types of c-elements using an LUT-based implementation.

Further, in the case of the bounded delay model, it requires a worst-case analysis for each data path to match the *Req* signal. Therefore, design-specific delay cells were also implemented by creating an LUT-based gate chain [19].

B. INTERFACING ASYNCHRONOUS CIRCUIT WITH SYNCHRONOUS BLOCK MEMORY

In an asynchronous circuit design, the interface with memory is one of the major design issues. Because most FPGAs provide memory cells in the form of synchronous block memory, an interface logic is required for asynchronous circuits in FPGAs to use this memory. The interface logic should be able to provide capabilities to generate the local clock and read/write control signals to the block memory and identify the response signals and data from the memory at the correct time.

Fig. 4 (a) shows the proposed interfacing module between the block memory and asynchronous core. The input clock for the block memory (*Mem_Clk*) is generated by read or write request signals (*Read_req*, *Write_req*) from the asynchronous core. These signals are aligned with the data and address changes from the data path, as shown in Fig. 4 (b). From the perspective of the data path, designers can reuse the data path of a synchronous core without changing to that of the asynchronous core because the targeted asynchronous core adopts a bounded delay model.

To guarantee the validity of the data from the block memory, the *Read_ack* and *Write_ack* signals are delayed by the time matched with the worst case of the block memory from when *Read_req* and *Write_Req* are asserted, respectively. The worst-case delays of read (*Twm_r*) and write (*Twm_w*) operations in block memory, as shown in the timing diagram in Fig. 4 (b), are implemented in LUT-based matched delay cells (*LUT_delay*). Hence, the functionality of the data path is dominated by the 4-phase handshake protocol, and the control path can be guaranteed concurrently.

Accordingly, the logic size of the proposed interfacing module between the AFSM-based controller and the block memory is relatively small. Thus, it did not induce performance degradation owing to the interfacing module. Only one *OR* gate delay was added, and the latency caused by the *LUT_delay* was initially hidden at the execution time of the block memory.

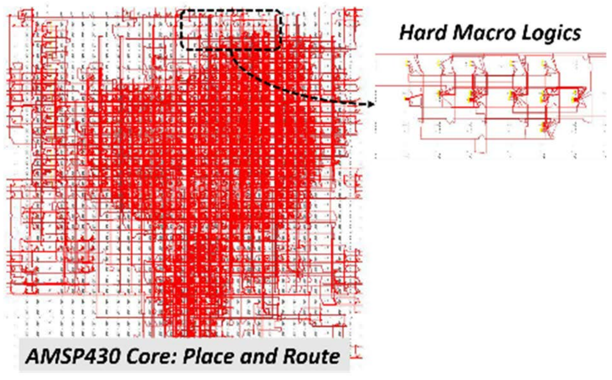


FIGURE 5. Hard macro on Xilinx Spartan3 (XC3S400) FPGA.

C. HARD MACRO-BASED ASYNCHORNOUS CONTROLLER IMPLEMENTATION

The optimization process normally changes the placement and routing (P&R) result according to modifications in the design information, such as the logic size and types of resources used, with an optimization strategy for the design tool. However, because the changes in the P&R result are related to timing, this optimization process can incur a timing difference between the matched delay cells and the respective data path modules implementing an asynchronous controller. In the case of a timing-critical control signal in the asynchronous controller, as it is sensitive to timing changes due to P&R, the location of the logic gates and wires of the timing-critical control signal must be fixed in the P&R process in the FPGA.

A hard macro-based design is a method that can maintain P&R. As the designer directly executes the P&R in small module units, it is possible to design an asynchronous controller optimally and maintain its timing, which is one of the constraints in asynchronous controller implementation in FPGA. Designers generate hard macro cells using an FPGA schematic editing tool during the mapping process after the synthesis.

In general, when the targeted design has a relatively larger size of hard macro blocks, from the perspective of the implementation time, it could have benefited [20]. However, the functionality and optimization of implemented design with hard macro blocks are guaranteed by the designer rather than a design tool.

The left part of Fig. 5 shows the P&R results of the asynchronous MSP430 core, which is explained in the next section using hard macro-based logic provided by the Xilinx FPGA design tool. As shown on the right side of Fig. 5, the timing-critical modules are implemented using hard-macro logic to fix the P&R of the internal logic. Otherwise, in the case of other modules that are not implemented by the hard-macro, the P&R results are changed.

IV. MSP430 ARCHITECTURE AND IMPLIMENTATION

MSP430 is a 16-bit low-power processor for sensing and measurement applications [21], [22], [23]. To design an

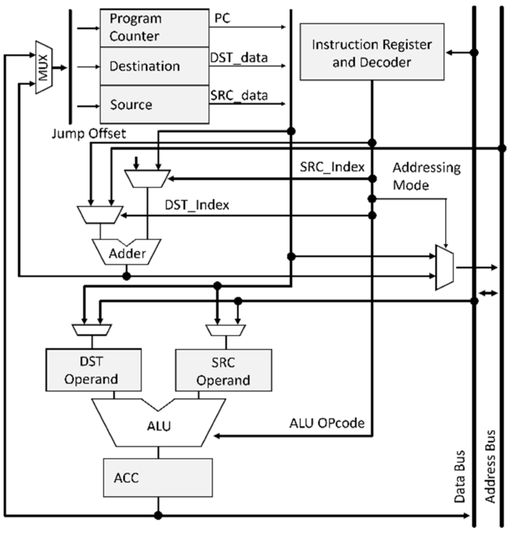


FIGURE 6. Data path for AMSP430.

MSP430 in an asynchronous circuit design methodology on FPGAs, we apply the design techniques addressed in Section III. This section describes the architecture of the asynchronous MSP430 (AMSP430) and its implementation on a commercial Xilinx FPGA.

A. ASYNCHRONOUS MSP430 ARCHITECTURE

The instruction set architecture of MSP430 has 27 core instructions and supports 24 emulated instructions [9]. The core instructions have a dedicated opcode that is directly decoded by the core. The core instructions are categorized into three groups according to the number of operands: dual-operand, single-operand, and jumps. In addition, MSP430 supports seven addressing modes, and all instructions can be used in any of the seven addressing modes without any restrictions. This flexible addressing mode is advantageous in the edge device environment because restrictions on the program memory size are frequent in the application for the edge device sphere. Depicted in Fig. 6, depending on the addressing mode for a given instruction, the instruction can flow flexibly to different data paths. Hence, it is more efficient to design a single pipelined data path rather than a multistage pipelined architecture.

Fig. 7 shows the architecture of the proposed AMSP430. The left part of the figure shows the designed data path of AMSP430. The control path shown on the right side of Fig. 7 is based on the AFSM [24] with 4-phase signaling. The control path consists of an IFID module for instruction fetching and decoding, an OF1 module for fetching the source operands, an OF2 module for fetching the destination operands, a JUMP module for processing the jump instructions, and an EXWB module for executing the instruction and interfacing memory.

To apply the asynchronous design methodology based on the bounded delay model, each control module has a delay element on the path from the *Req* signal to the *Ack* signal

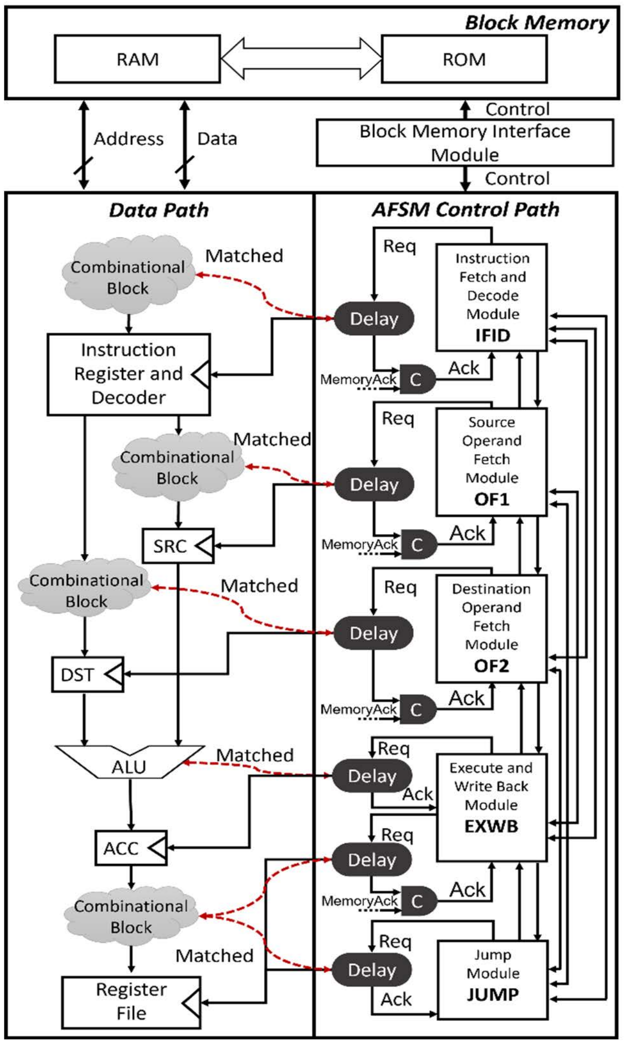

Block Memory
RAM
ROM
Address
Data
Control
Block Memory Interface Module
Control
Data Path
AFSM Control Path
Combinational Block
Matched
Instruction Register and Decoder
SRC
DST
ALU
ACC
Register File
Delay
Req
Ack
MemoryAck
C
Instruction Fetch and Decode Module IFID
Source Operand Fetch Module OF1
Destination Operand Fetch Module OF2
Execute and Write Back Module EXWB
Jump Module JUMP

FIGURE 7. Proposed architecture of AMSP430.

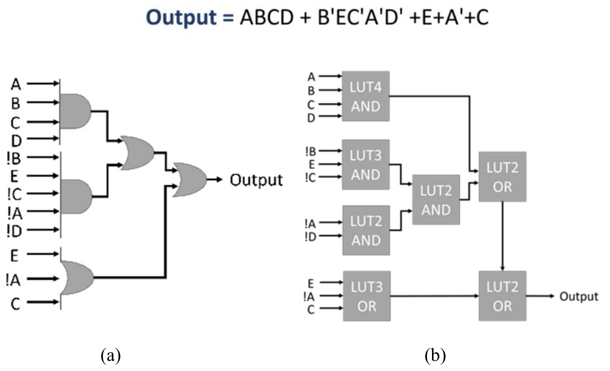


FIGURE 8. Boolean logic and LUT mapping structure. (a) Example of boolean equation for AFSM controller and logic gate schematic and (b) LUT mapping structure.

that is matched with the time for the critical path in each combinational block in the data path.

Furthermore, to execute the application on the core, the block memory interworks with the designed data path and the AFSM-based control path for passing the data (operation code) and controlling the memory interface timing described in Section III. B, respectively. This memory interface module also has a handshake signal. Therefore, the IFID, OF1, OF2, and EXWB modules, which cooperate with block memory, should synchronize the control signal between other control modules and the memory interface module. Consequently, the c-elements should be implemented in the designed control path to synchronize the *Ack* signal locally.

B. IMPLEMENTATION OF AMSP430 ON FPGA

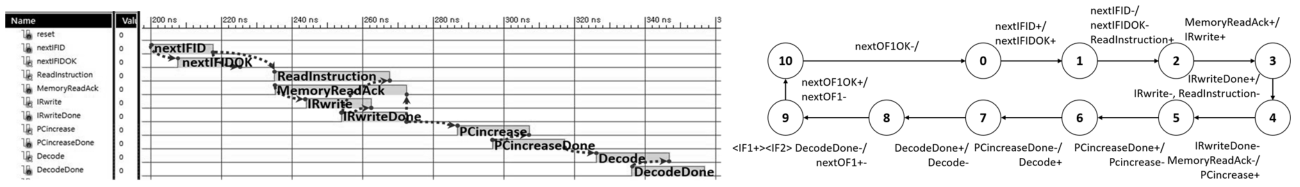
The AMSP430 core and block memories shown in Fig. 7 were implemented in commercial Xilinx FPGAs (Spartan3: XC3S400 [25] and Virtex-5: XC5VLX110T [26]).

The data path for AMSP430 was implemented using normal synchronous circuit design methodology. The AFSM-based control modules were synthesized using the gate-level Boolean equation with a 3D [11] tool and their outputs were implemented by adapting the LUT-based coding style proposed in Section III. A, using an FPGA design environment (Xilinx ISE).

The delay elements in the control path were implemented using the method proposed in Section III. A. Additionally, some of the timing-critical control logics (e.g., internal feedback logics) in the control module were implemented using hard macro to ensure the timing of each signal, as described in Section III. C.

Fig. 8 shows an example of the implementation of a Boolean equation for an AFSM-based control module by applying LUT-based designs. Fig. 8 (a) shows the 1:1 mapping for the Boolean equation from the 3D tool to the gate level. With LUT-based designs, the equation of a gate with multiple inputs can be implemented by breaking it down into LUTs with few inputs, as shown in Fig. 8 (b).

Each control module was also verified through a simulation tool for FPGA, Xilinx ISim, at the post P&R level. Fig. 9 (a) shows the captured waveforms, and Fig. 9 (b) shows the control flows of the IFID module based on the AFSM controller with 4-phase signaling. Fig. 10 represents the abstracted level of the delay path for the IFID module that has five different paths. As shown in Fig. 9 (a) and (b), a *nextIFID* signal, which is generated by the EXWB module, is excited to fetch the instruction from the memory. Subsequently, as shown in Fig. 10, after around 8 ns, a *nextIFIDOK* signal is asserted as an *ACK* to notify the arrival of a control signal. Then, the falling *nextIFID* signal causes the *readinstruction* signal to rise. Subsequently, a *MemoryReadAck* signal is propagated from the memory interface around 30 ns (the maximum delay of memory block) later to represent the arrival of instruction at the core. This *MemoryReadAck* signal causes the controller to generate an instruction register write signal, which is *IRwrite*. The IFID module then generates signals to increase the program counter (PC) and decode the fetched instructions. Accessing the register files takes around 10 ns, as described in Fig. 10. Finally, according to the given instruction, the IFID module generates OF1 Req or OF2 Req to invoke the



(a) Waveform capture (b) AFSM Control flow

FIGURE 9. IFID partial simulation waveform capture & AFSM control flow.

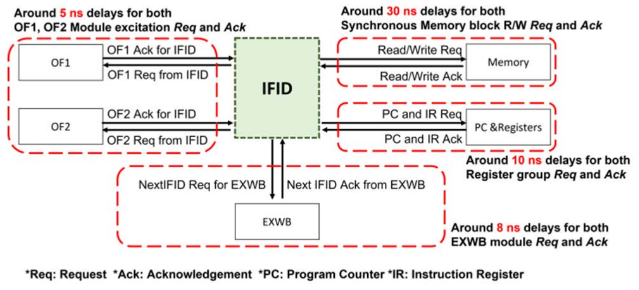

Around 5 ns delays for both OF1, OF2 Module excitation Req and Ack
Around 30 ns delays for both Synchronous Memory block R/W Req and Ack
OF1
OF1 Ack for IFID
OF1 Req from IFID
OF2
OF2 Ack for IFID
OF2 Req from IFID
IFID
Read/Write Req
Read/Write Ack
Memory
PC and IR Req
PC and IR Ack
PC &Registers
Around 10 ns delays for both Register group Req and Ack
NextIFID Req for EXWB
Next IFID Ack from EXWB
EXWB
Around 8 ns delays for both EXWB module Req and Ack
*Req: Request *Ack: Acknowledgement *PC: Program Counter *IR: Instruction Register

FIGURE 10. Abstracted level of delay path for the IFID module.

TABLE 1. Memory map.

Purpose	Address Range
RESET	0xFFFE – 0xFFFF
Non-Masking Interrupt (NMI)	0xFFFC – 0xFFFE
TIMER	0xFFF0 – 0xFFF5
Input/Output(I/O)	0xFFE4 – 0xFFE7
User Program(TEXT)	0xFE00 – 0xFE32
STACK	0x024C – 0x027E

respective modules. These invoking processes take around 5 ns as shown in Fig. 10.

C. INTERFACING WITH COMPILER

MSP430 provides open compiling environments. Utilization of the asynchronous core implemented in FPGA at the system level necessitates the analysis of the memory map, bootloader, and header file of a commercial MSP430.

Table 1 contains the memory map created by Texas Instruments CSS v.7.1 with an MSP430 Linker v.16.9.1 [27], which targets the MSP430G2001 version. The provision of congeniality with the implemented asynchronous core and commercial compiler requires a memory map for the block memory to be configured, as shown in Table 1.

In our system, the address from 16'hFE00 to 16'hFE32 is dedicated to read-only memory (ROM) for a user application. Because our system is implemented on an FPGA with block memory, the application is initialized during the synthesis process using the Xilinx Core Generator.

Commercialized MSP430 typically has a bootloader for user application fetching and initialization of the device. However, in our design, as the user application can be stored in block memory, during the implementation process, a specified boot sequence is not required. Therefore, we directly write the application binary code into the ROM and RAM to ensure functional compatibility. The application binary code includes the header file, which contains the symbolic address of each I/O pin in the application code from the output of the commercial compiler.

V. EXPERIMENTS AND COMPARISONS

In this section, we address the experimental results of running an actual application on the AMSP430, which was implemented above, and its synchronous counterpart (SMSP430). Furthermore, the experimental results also cover the reliability of each group under an unstable voltage condition that is faced by numerous edge devices. Finally, we describe the measurement results for each experimental group of electromagnetic wave emission characteristics, which is an emerging concern in wearable devices.

A. RUNNING A REAL APPLICATION ON THE ASYNCHRONOUS MSP430

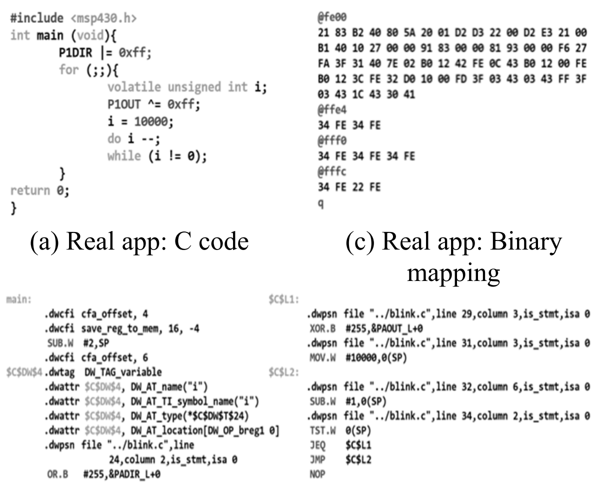
To verify the interworking between a commercial compiler and AMSP430, the timer program, which is frequently used in edge devices, was emulated, as shown in Fig. 11 (a). The codes in Fig. 11 (a) are translated to assembly code in Fig. 11 (b) by the MSP430 assembler, and compiled into a binary code as represented in Fig. 11 (c) using TI CSS v.7.1. The binary code is directly input to the block memory in AMSP430.

To monitor the functionalities of the timer application on the AMSP430 in the Xilinx Virtex-5 evaluation board, we mapped the I/O from the application code to the data output. The data output from the core is connected to an LED on the evaluation board to act as an indicator. If the timer application is properly operated, the indicator blinks during the specified period.

As shown in Fig. 12, the AMSP430 indicator blinks periodically, according to the sensing time determined in the application code. In conclusion, by performing this experiment, we confirmed the congeniality between AMSP430 and a commercial compiler.

B. EXPERIMENTATION AND COMPARISON OF AMSP430 AND SMSP430

Based on the feasibility of AMSP430 on a Xilinx FPGA as shown in the previous section, we evaluated its operational



```
#include <msp430.h>
int main (void){
        P1DIR |= 0xff;
        for (;;){
                volatile unsigned int i;
                P1OUT ^= 0xff;
                i = 10000;
                do i --;
                while (i != 0);
        }
return 0;
}
```

(a) Real app: C code

```
@fe00
21 83 B2 40 80 5A 20 01 D2 D3 22 00 D2 E3 21 00
B1 40 10 27 00 00 91 83 00 00 81 93 00 00 F6 27
FA 3F 31 40 7E 02 B0 12 42 FE 0C 43 B0 12 00 FE
B0 12 3C FE 32 D0 10 00 FD 3F 03 43 03 43 FF 3F
03 43 1C 43 30 41
@ffe4
34 FE 34 FE
@fff0
34 FE 34 FE 34 FE
@fffc
34 FE 22 FE
q
```

(c) Real app: Binary mapping

```
main:
        .dwcfi  cfa_offset, 4
        .dwcfi  save_reg_to_mem, 16, -4
         SUB.W  #2,SP
        .dwcfi  cfa_offset, 6
$C$DW$4.dwtag  DW_TAG_variable
        .dwattr $C$DW$4, DW_AT_name("i")
        .dwattr $C$DW$4, DW_AT_TI_symbol_name("i")
        .dwattr $C$DW$4, DW_AT_type(*$C$DW$T$24)
        .dwattr $C$DW$4, DW_AT_location[DW_OP_breg1 0]
        .dwpsn  file "../blink.c",line
                    24,column 2,is_stmt,isa 0
         OR.B   #255,&PADIR_L+0
```

```
$C$L1:
        .dwpsn  file "../blink.c",line 29,column 3,is_stmt,isa 0
         XOR.B  #255,&PAOUT_L+0
        .dwpsn  file "../blink.c",line 31,column 3,is_stmt,isa 0
         MOV.W  #10000,0(SP)
$C$L2:
        .dwpsn  file "../blink.c",line 32,column 6,is_stmt,isa 0
         SUB.W  #1,0(SP)
        .dwpsn  file "../blink.c",line 34,column 2,is_stmt,isa 0
         TST.W  0(SP)
         JEQ    $C$L1
         JMP    $C$L2
         NOP
```

(b) Real app: Assembly mapping

FIGURE 11. Real application code.

stability in an unstable supply voltage scenario and with low power consumption characteristics by comparing it with an SMSP430 version.

To verify the functionality of both AMSP430 and SMSP430 on FPGA, we modeled the behavioral model of the MSP430 core, which was executed on an AVR XMEGA processor [28], to extract the reference data for both versions. The extracted reference data included the operation codes described in internal register values, and timing information of the interface signals. Using the reference data, we monitored the operation code as data input for each version of the block memory.

We also checked the continuous changes in the value in the program counters of each version as an address bus, according to the given operation code. Therefore, in all cases, it was confirmed that the behavior of the two versions and that of the reference model were identical.

Fig. 13 shows the experimentation environment for verifying the performance of AMSP430 and SMSP430 under an unstable supply voltage scenario. An oscilloscope was used to monitor the instruction and data fetch cycle to measure the performance of both versions.

A power supply was used to change the input voltage into the targeted evaluation board. The AVR was used for verifying the functionality of the targeted system. In these experiments, the benchmark program [29] was executed on the Xilinx Spartan3 (XC3S400) FPGA evaluation board for both AMSP430 and SMSP430.

Fig. 14 depicts the operational frequency curves for both AMSP430 and SMPS430 depending on the unstable input power, which was less than the normal supply voltage of 5 V.

When the targeted system passed the functional verification, which was performed by AVR, the average frequency of the end signal at the EXWB stage was verified with an oscilloscope, with the supply voltage decreasing by 0.2 V from 5 V.

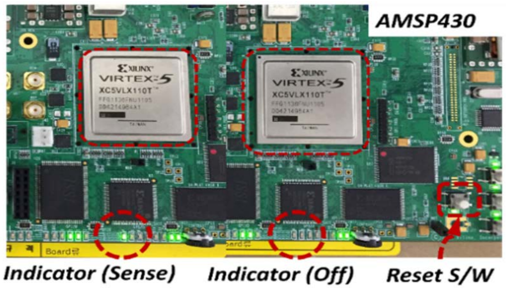


FIGURE 12. Program running result on the Xilinx Virtex-5(XC5VLX110T) FPGA.

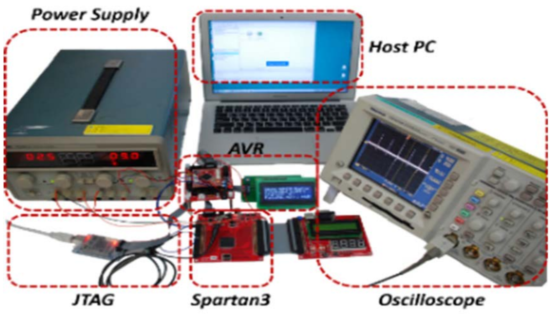


FIGURE 13. Experimentation environment.

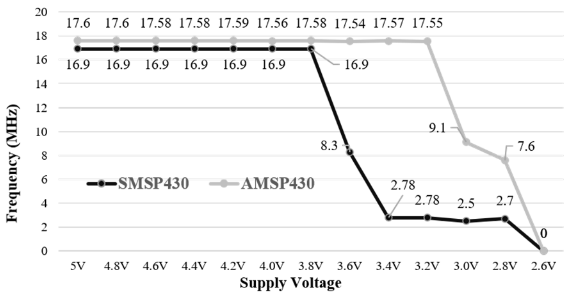


FIGURE 14. Operational frequency.

Through this experiment, we observed that AMSP430 and SMSP430 maintain their performance at 17.6 MHz and 16.9 MHz, respectively, up to 3.8 V.

When the supply voltage drops below 3.8 V, the operational frequency of SMSP430 degrades significantly. This is because the digital clock manager (DCM) [30], a circuit for generating a global clock inside the FPGA, malfunctions owing to a low operating voltage.

AMSP430 operates at maximum performance up to a supply voltage of 3.2 V. In addition, AMSP430 shows a 2.8 to 6.3 times better performance than SMSP430 even when the supply voltage decreases from 3.2 V to 2.8 V. The reason for this result is that AMSP430 does not have any globalized control signal such as a clock, and hence, DCM cannot affect the functionality of the data path and control path in AMSP430. Thus, under unstable power supply conditions, AMSP430 had more fault-tolerant capability than SMSP430.

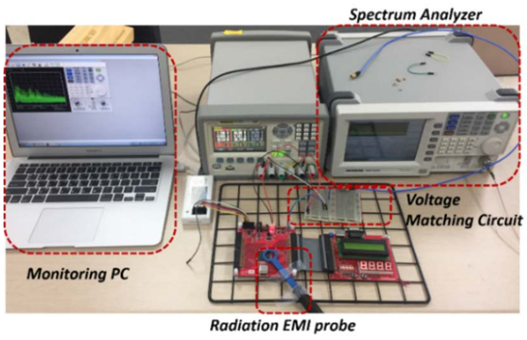
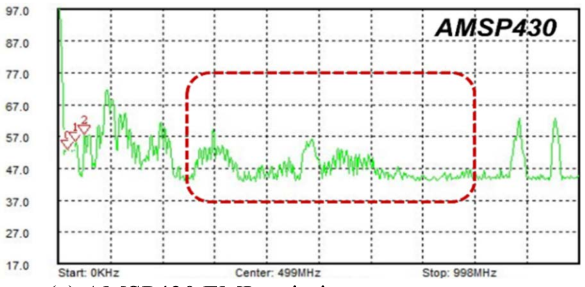
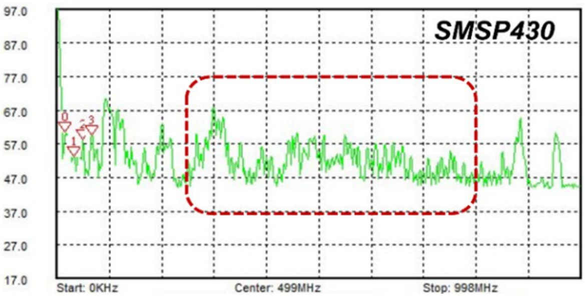


FIGURE 15. Environment for the EMI measurement.



(a) AMSP430 EMI emission spectrum capture.



(b) SMSP430 EMI emission spectrum capture.

FIGURE 16. EMI emission spectrum.

We also obtained more detailed information on the power consumption of AMSP430 and SMP430 by performing a power simulation using Xilinx ISim [31]. The power simulation revealed that AMSP430 and SMSP430 consume power of 59.79mW and 97.04mW, respectively, both under typical conditions. This means that AMSP430 consumes 62.3% less power than SMSP430. Because AMSP430 does not have a globally distributed clock network, it can have a lower switching power than SMSP430.

C. MEASUREMENT OF ELECTROMAGNETIC INTERFERENCE AND COMPARISONS

As edge devices evolve into wearable devices, there are a number of issues regarding Electromagnetic Interference (EMI) that can affect the human body [32]. Nonetheless, because most commercialized edge devices are designed as synchronous systems, the platform generates EMI peaks pursuant to the global and periodic control signal, that is, a clock signal. In contrast, asynchronous processors have the advantage of superior EMI emission characteristics compared with synchronous processors because the global clock is fundamentally eliminated.

We measured EMI emission, which is normally measured on a supply voltage pin to measure the power-ground noise, by setting up the test environment as shown in Fig. 15 and AMSP430 and SMSP430 operated at 17.6 MHz and 16.9 MHz, respectively, at 5 V.

Fig. 16 presents the results of the EMI emission measurements. The maximum EMI emission of AMSP430 was measured to be 52.7 dBuV, which was less than the 60.3 dBuV of SMSP430. Moreover, the EMI emission characteristic of AMSP430 also shows that in the 250 MHz area (the region inside the red box in Fig. 16), the EMI emission is significantly degraded compared to SMSP430.

This means that if a wireless edge device operating at a specific frequency exists, a peripheral circuit designed asynchronously would emit fewer electromagnetic waves that cause malfunctions at the frequency of the wireless device than a peripheral circuit designed synchronously.

VI. CONCLUSION AND FUTURE WORK

In this study, we proposed design techniques for an AFSM controller in a commercial FPGA to overcome the drawbacks of conventional synchronous circuit designs for the applications of edge devices. To verify the proposed design techniques, we designed and implemented asynchronous MSP430 for a platform for an edge device. In addition, we designed an interfacing module between the asynchronous core and the synchronous block memory. We also analyzed the commercial compiler and adapted its results to achieve congeniality with the implemented core and the commercial MSP430.

In an experiment that assumes the running environment of an edge device, the external input power is not usually stable at 5 V.

Under such conditions, the asynchronous MSP430 was found to function until the supply voltage dropped to 3.0 V. Conversely, the synchronous MSP430 started to malfunction at 3.6 V. Moreover, the asynchronous MSP430 was found to consume 62.3% less power than its synchronous counterpart. The EMI measurement also indicated that at working frequency, the asynchronous MSP430 emits 7.6 dBuV less EMI than the synchronous MSP430.

In the future, to provide functionality equivalent to that of commercial MSP430, various I/O devices, and their interfacing modules will be added.

Moreover, to verify the scalability of the proposed design techniques, an asynchronous RISC-V [33] core, for edge devices, will also be designed and implemented.

REFERENCES

[1] Y. I. Ismail, "Interconnect design and limitations in nanoscale technologies," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2008, pp. 780–783, doi: 10.1109/ISCAS.2008.4541534.

[2] C. J. Anderson, J. Petrovick, J. M. Keaty, J. Warnock, G. Nussbaum, J. M. Tendier, C. Carter, S. Chu, J. Clabes, J. DiLullo, and P. Dudley, ''Physical design of a fourth-generation POWER GHz microprocessor,'' in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2001, pp. 232–233, doi: 10.1109/ISSCC.2001.912617.

[3] *Intel Ethernet Switch FM5000/6000 Datasheet*, Netw. Division (ND), Intel. Mountain View, CA, USA, 2017, pp. 15–19.

[4] (2022). *TrueNorth*. [Online]. Available: https://www.ibm.com/blogs/research/tag/truenorth/

[5] R. Paz-Vicente, E. Cerezuela-Escudero, M. Dominguez-Morales, A. Jimenez-Fernandez, A. Linares-Barranco, and G. Jimenez-Moreno, ''A performance comparison study between synchronous and asynchronous FPGA for spike based systems. Under the AER synthetic generation,'' in *Proc. Int. Symp. Perform. Eval. Comput. Telecommun. Syst.*, 2011, pp. 38–45.

[6] (2022). *Achronix Semiconductor Corporation*. [Online]. Available: https://www.achronix.com/

[7] L. Yakovlev and A. Sangiovanni-Vincentelli, ''A unified signal transition graph model for asynchronous control circuit synthesis,'' in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 1992, pp. 104–111, doi: 10.1109/ICCAD.1992.279390.

[8] C. J. Myers, ''Graphical representations,'' in *Asynchronous Circuit Design*. Hoboken, NJ, USA: Wiley, 2001, pp. 85–130, doi: 10.1002/0471224146.ch4.

[9] *MSP430x2xx Family User's Guide*, Texas Instrument, Dallas, TX, USA, 2013.

[10] J. Sparso and S. Fuber, ''Fundamentals,'' in *Principles of Asynchronous Circuit Design—A System Perspective*. New York, NY, USA: Springer, 2001, pp. 9–28.

[11] K. Y. Yun, ''Synthesis of asynchronous controllers for heterogeneous systems,'' Ph.D. dissertation, Dept. Elect. Eng., Stanford Univ., Stanford, CA, USA, 1994.

[12] D. L. Oliveira, S. S. Sato, O. Saotome, and R. T. de Carvalho, ''Hazard-free implementation of the extended burst-mode asynchronous controllers in look-up table based FPGA,'' in *Proc. 4th Southern Conf. Program. Log.*, Mar. 2008, pp. 143–148, doi: 10.1109/SPL.2008.4547746.

[13] F. T. D. F. Barbosa, D. L. De Oliveira, T. S. Curtinhas, L. D. A. Faria, and J. F. D. S. Luciano, ''Implementation of locally-clocked XBM state machines on FPGAs using synchronous CAD tools,'' *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 5, pp. 1064–1074, May 2017, doi: 10.1109/TCSI.2017.2649102.

[14] M. M. Kim and P. Beckett, ''Design techniques for NCL-based asynchronous circuits on commercial FPGA,'' in *Proc. 17th Euromicro Conf. Digit. Syst. Design*, Aug. 2014, pp. 451–458, doi: 10.1109/DSD.2014.85.

[15] Z. Xia, S. Ishihara, M. Hariyama, and M. Kameyama, ''An asynchronous FPGA based on dual/single-rail hybrid architecture,'' in *Proc. Int. Conf. Eng. Reconfigurable Syst. Algorithms (ERSA), Steering Committee World Congr. Comput. Sci., Comput. Eng. Appl. Comput.*, 2012, pp. 139–142.

[16] Y. Liu, G. Xie, P. Chen, J. Chen, and Z. Li, ''Designing an asynchronous FPGA processor for low-power sensor networks,'' in *Proc. Int. Symp. Signals, Circuits Syst.*, Jul. 2009, pp. 1–6, doi: 10.1109/ISSCS.2009.5206091.

[17] R. H. B. Netzer and B. P. Miller, ''What are race conditions? Some issues and formalizations,'' *ACM Lett. Program. Lang. Syst.*, vol. 1, no. 1, pp. 74–88, Mar. 1992, doi: 10.1145/130616.130623.

[18] Q. T. Ho, J. B. Rigaud, L. Fesquet, M. Renaudin, and R. Rolland, ''Implementing asynchronous circuits on LUT based FPGAs,'' in *Proc. Int. Conf. Field Program. Log. Appl.* Berlin, Germany: Springer, 2002, pp. 36–46.

[19] Z. Shin, M. H. Oh, H. Kwon, H. Kim, and D. Kang, ''Implementation of an asynchronous micro-controller on the commercial FPGA,'' *Int. J. Comput. Theory Eng.*, vol. 9, no. 6, pp. 466–472, Dec. 2017, doi: 10.7763/IJCTE.2017.V9.1188.

[20] C. Lavin, B. Nelson, and B. Hutchings, ''Impact of hard macro size on FPGA clock rate and place/route time,'' in *Proc. 23rd Int. Conf. Field Program. Log. Appl.*, Sep. 2013, pp. 1–6, doi: 10.1109/FPL.2013.6645510.

[21] A. I. Abdul-Rahman and C. A. Graves, ''Internet of Things application using tethered MSP430 to thingspeak cloud,'' in *Proc. IEEE Symp. Service-Oriented Syst. Eng. (SOSE)*, Mar. 2016, pp. 352–357, doi: 10.1109/SOSE.2016.42.

[22] A. Roy, P. J. Grossmann, S. A. Vitale, and B. H. Calhoun, ''A 1.3 μW, 5pJ/cycle sub-threshold MSP430 processor in 90 nm xLP FDSOI for energy-efficient IoT applications,'' in *Proc. 17th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2016, pp. 158–162, doi: 10.1109/ISQED.2016.7479193.

[23] D. Thomas, R. McPherson, G. Paul, and J. Irvine, ''Optimizing power consumption of Wi-Fi for IoT devices: An MSP430 processor and an ESP-03 chip provide a power-efficient solution,'' *IEEE Consum. Electron. Mag.*, vol. 5, no. 4, pp. 92–100, Oct. 2016, doi: 10.1109/MCE.2016.2590148.

[24] Z. Shin, M.-H. Oh, J.-G. Lee, H. Y. Kim, and Y. W. Kim, ''Design of a clockless MSP430 core using mixed asynchronous design flow,'' *IEICE Electron. Exp.*, vol. 14, no. 8, 2017, Art. no. 20170162, doi: 10.1587/elex.14.20170162.

[25] Xilinx. (Jun. 27, 2013). *Spartan-3 FPGA Family Data Sheet*. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds099.pdf

[26] Xilinx. (Mar. 16, 2012). *Virtex-5 FPGA User Guide*. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug190.pdf

[27] *Code Composer Studio (CSS) Integrated Development Environment (IDE)*. Texas Instruments. Accessed: Oct. 25, 2022. [Online]. Available: http://www.ti.com/tool/CCSTUDIO

[28] Microchip. *8-Bit AVR MCUs*. Accessed: Oct. 25, 2022. [Online]. Available: https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/avr-mcus

[29] L. Nazhandali, M. Minuth, and T. Austin, ''SenseBench: Toward an accurate evaluation of sensor network processors,'' in *Proc. IEEE Int. Workload Characterization Symp.*, Oct. 2005, pp. 197–203, doi: 10.1109/IISWC.2005.1526017.

[30] Xilinx. (Jan. 5, 2006). *Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs*. [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp462.pdf

[31] Xilinx. (Apr. 24, 2012). *ISim User Guide*. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documentation/sw_manuals/xilinx14_1/plugin_ism.pdf

[32] P. Phunchongharn, E. Hossain, D. Niyato, and S. Camorlinga, ''A cognitive radio system for E-health applications in a hospital environment,'' *IEEE Wireless Commun.*, vol. 17, no. 1, pp. 20–28, Feb. 2010, doi: 10.1109/MWC.2010.5416346.

[33] RISC-V. *About RISC-V*. Accessed: Oct. 25, 2022. [Online]. Available: https://riscv.org/about/

ZIHO SHIN received the B.E. degree in electrical and electronics engineering from the University of Ulsan, Ulsan, South Korea, in 2015, and the M.E. degree in computer software from the Electronics and Telecommunication Research Institute (ETRI) Campus, University of Science and Technology (UST), Daejeon, South Korea, in 2018. He has been with The Affiliated Institute of ETRI, since December 2017. His research interests include asynchronous circuit design, high-performance computing systems, high-speed system fabric interconnection design, and cryptographic hardware design.

MYEONG-HOON OH received the Ph.D. degree in information and communications engineering from the Gwangju Institute of Science and Technology (GIST), Gwangju, South Korea, in 2005. He was a Principal Researcher at the Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea, from 2005 to 2021. During the same period, he was an Associate Professor at the University of Science and Technology (UST), Daejeon. He has been an Assistant Professor at Honam University, Gwangju, since 2021. His current research interests include high-performance computing systems, cloud-computing infrastructure, and cloud-computing standardization. He has been an Editor for cloud computing at ITU-T SG13, since 2012.