논문 19-44-06-23 The Journal of Korean Institute of Communications and Information Sciences '19-06 Vol.44 No.06
https://doi.org/10.7840/kics.2019.44.6.1193

IoT Things를 위한, 설계변경이 용이한 실용적 HW/SW 설계 방법

배 점 한◆, 김 종 태°

Practical HW/SW Design Method for Easy Design Change for IoT Things

Jum-Han Bae◆, Jong Tae Kim°

요 약

IoT 시스템은, 사물이 인터넷에 접속되어 동작하게 되는 구조 이다. 사물은 비교적 간단한 하드웨어(HW)와 소프트웨어(SW)로 구성되어 있고, 동작 속도는 비교적 느리다. 그러나 하드웨어에서부터 상위 어플리케이션 계층까지 넓은 범위를 설계를 해야 한다. 본 논문은 IoT 시스템에서 다양한 사물(Things)부분의 실용적인 설계 방법을 다룬 튜토리얼 논문이다. 사물(Things) 부분은 센서와 하드웨어와 소프트 웨어로 구성 되며, 와이파이(WiFI)기능을 갖는 허브(Hub)를 통해서 인터넷에 접속하는 것으로 가정한다. 허브 부분은 상용의 제품으로 되어 있고, 센서는 부품으로 구성되어 있기 때문에 사물부분에서의 하드웨어와 소프트웨어를 설계하는 방법에 초점을 두었다. 사물(Things) 부분은 소량이면서도 다양한 종류가 있어야 한다. 그렇기 때문에 하드웨어의 설계에서도 자유도를 가질 수 있도록 FPGA(Field Programmable Gate Array)를 사용하였다. 또한 동작 속도가 느리지만 쉬운 프로그래밍(Programming)이 가능한 아두이노(Arduino) 보드를 사용하여 소프트웨어를 설계하였다. 이러한 방법은 시간, 노력, 사용 부품, 투자 등의 비용을 최소로 하면서도 높은 설계 자유도를 가질 수 있도록 할 수 있다. 이것은 IoT 시스템의 특성을 활용하기 때문에 가능한 것이다.

Key Words : IoT system, FPGA, Things, Arduino,Tutoral

ABSTRACT

The IoT system is a structure in which objects are connected to the Internet and operated. Things are composed of relatively simple hardware (HW) and software (SW), and the operating speed is relatively slow. However, it is necessary to design a wide range from hardware to higher application layer. This paper is a tutorial paper on practical design methods of various parts of Things in IoT system. The Things section consists of sensors, hardware and software, and is connected to the Internet via a hub with WiFi capability. Because the hub is a commercial product, and the sense is made up of parts, I focused on how to design hardware and software in things. Things have a small amount but a wide variety of things. For this reason, Field Programmable Gate Array (FPGA) is used to allow freedom in hardware design. We also designed the software using the Arduino board, which is easy to program with a slow operation speed. Such a method can have a high degree of design freedom while minimizing the cost of time, effort, parts used, and investment. This is possible because it exploits the characteristics of the IoT system.

◆ First Author : Soongsil University, Department of IT Convergence, jhabe17@ssu.ac.kr,정회원
/ Electronic, Electrical and Computer Eng., Doctoral Program, Sungkyunkwan Univ., Republic of Korea
° Author : Sungkyunkwan University, School of Electronic and Electrical Engineering. jtkim@skku.edu, 종신회원
논문번호 : 201903-009-D-RE, Accepted March 6, 2019; Revised April 2, 2019; Accepted April 2, 2019

Ⅰ. 서 론

IoT (Internet of Things) 시스템은 사물과 인터넷이 융합되어 있다. 하위 하드웨어의 설계에서부터 어플리케이션 설계까지를 하여야 하고, 다품종 소량생산의 다양성이 있어서 일괄적으로 양산된 하드웨어로는 모든 경우의 대응이 되지 않는다. 설계의 자유도를 가지면서도 IoT의 특성과 한계를 만족하도록 하는 방법을 적용하여야 한다. 방법은 현실성에 바탕을 둔 타당한 방법이어야 한다. 쉽게 구입이 가능하고 비용을 최소로 하는 방법이어야 한다. 비용은 부품의 구입비용만이 아니라, 설계 노력비용, 설계환경 투자비용, 확장성 등을 고려해서 정해야 한다. IoT 시스템에서 사물(Things)의 특징인 소량 다품종의 다양성을 만족해 주기 위해서는 사용할 디바이스(Device)는 소량 구입에 제약이 없고, 프로그램어블(Programmable) 해야 한다. SoC 칩은 설계, 제조에 많은 시간과 비용을 필요로 한다. 그리고 IoT 시스템은 인터넷에 접속하여 데이터를 주고받는 방식이기 때문에 대부분의 IoT 응용에서 사물부분은 빨리 동작할 필요가 없다. 또한 IoT는 물리계층에서 응용계층까지 다양한 설계를 해하기 때문에 쉽게 설계 할 수 있는 개발환경이 되어야 한다. 인터넷에 접속하고 클라우드(Cloud)를 통한 원격제어에 대해서는 앞서 발표된 논문인 "ARTIK 클라우드를 통한 분산된 원격 IoT 센서 데이터 모니터링 및 제어"[1]를 통하여 기술되어 있기 때문에 여기서는 사물분야에 집중하도록 한다.

IoT 관련하여 많은 설계가 되고 있는 시점은 최근이다. 이전에는 FPGA나 아두이노 보드는 검증을 하기 위한 중간단계로 사용되었다. 생산 대수가 많지 않는 통신용 기지국이나 산업용 장비 등에서 사용되고 있다. 이런 경우는 성능이 최우선시 되고, 최적화 설계나 저비용 설계는 큰 고려 대상이 아니다. 이런 영역은 FPGA의 느린동작 속도가 문제가 된다. 본 논문에서는, 소규모 IoT 설계라는 관점에서 보았을 때 구현의 자유도, 구입비용, 설계 시간, 구입의 용이성 등의 문제를 해소해 보고자 하는 것이다. 또한 설계가 정확한지 검증하는 것도 주요한 이슈이다. 설계의 검증은 시뮬레이션(Simulation)을 하거나 실제 동작을 시켜서 동작의 정확성을 검증한다. FPGA는 사전에 시뮬레이션이 가능 하다. FPGA나 아두이노 보드는 코드가 완성되면 바로 실제 동작을 시켜 볼 수 있기 때문에 검증이 매우 편리하다. 이러한 방법론은 작은 단위의 시스템을 효과적으로 만들 수 있는 장점이다. 큰 시스템을 위한 방편으로는 시스템의 일부를 만들어서 다양한 검증을 해 보는 부분적인 용도로 사용 할 수 있다.

하드웨어는 FPGA 시장점유율 1위 회사인 자일링스(Xilinx)의 가장 저가 버전인 스파르탄6(Spartan6)[2]를 사용하고, 소프트웨어는 아두이노의 우노 버전[3]을 사용한다. 여기에 사용되는 통합설계환경(IDE)이나 논리 시뮬레이션(Simulation) 환경은 공식적으로 무료로 배포되고 있다. 구입비용에 있어서도, FPGA개발보드, FPGA 프로그램 다운로드, 아두이노 보드 등은 각각 수 만 원 정도의 저 비용이다. 그러나 IoT용 사물로서의 성능의 발휘에는 전혀 문제가 없을 수준으로 구현이 가능하다. 다른 FPGA를 사용할 수도 있고, 아두이노 대신 라즈베리 파이 보드를 사용 할 수도 있다. 프로그램어블(Programmable)한 하드웨어와 소프트웨어가 있으면 가능한 방법이다.

지금까지의 관련 사례들을 살펴보면, 기존 외출 시스템에서 저렴한 비용으로 유연한 무선 솔루션을 설계하고 구현하는 방법을 소개하기 위해서 저렴한 라즈베리 파이를 이용한 시스템이 소개 되었다.[4] 하드웨어는 개별 소자를 이용하여 구현 되었다. 또한 IT를 이용한 생육환경 모니터링 및 제어 플랫폼에 관한 연구가 있었다.[5] 아두이노와 개별 센스를 이용한 구성으로 되어 있다. 또 다른 분야로서 카메라와 같은 높은 성능과 OS를 요하는 곳에서는 라즈베리 파이를 이용하여 안면인식 기능을 실현한 논문이 있다.[6] 현재 시점에서는 하드웨어와 소프트웨어가 개별로 동작하는 상태이다. 만약 좀 더 융합적인 다양성을 요구하는 상태가 되면 IoT의 사물 부분도 하드웨어와 소프트웨어가 긴밀하게 융합된 상태로 발전해 가야 한다. 레지스터 테이블로 하드웨어를 제어 할 수 있으면 시스템 설계의 많은 유연성을 제공해 준다. 본 논문은 하드웨어와 소프트웨어가 융합된 플랫폼을 용이하게 구축하는 용도로 활용되고자 한다.

Ⅱ. 본 론

IoT 시스템은 센서 소자에서부터 인터넷 클라우드 서버까지의 많은 구성요소를 포함하고 있다. 구성요소가 많은 가운데서도 사물부분은 하드웨어의 비중이 크기 때문에 이 부분을 안정화 시켜야 만이 전체 시스템이 안정된다. 하드웨어는 변경과 재설계에 많은 시간과 노력이 들어가기 때문에 시행착오를 줄이는 것이 매우 중요하다. 사물부분의 구조도 유연성 있게 설계를 해 둔다면 사양 변화에 쉽게 대응할 수 있다.

2.1 기존의 SoC 구조

그림 1은 연결성을 기준으로 본 SoC의 구조 이다. 칩의 내부에는 시스템 버스(System Bus)가 있어서 칩 내부 블록끼리 연결해 준다. M#으로 표시된 부분은 마스터(Master) 이고 S#로 표시된 부분은 슬레이브(Slave)이다. 마스트는 능동적으로 버스에 연결요청을 할 수 있는 블록이고, 슬레이브는 요청이 오면 처리하는 수동적인 동작을 하는 부분이다. 주요 기능 블록들이나, CPU (Central Processing Unit), GPU (Graphics Processing Unit) 등이 마스터로 동작하고, 메모리 계열이 슬레이브로 동작한다.

그림 2는 마스터가 시스템 버스를 이용하는 사례를 보인 것이다. 시스템 버스는 어느 한 순간에는 특정 마스터의 데이터만 전달한다. 해당 마스트는 일정한 시간 주기로 버스를 이용하면서 해당 슬레이브와 데이터를 주고받는다. 모든 마스터와 모든 슬레이브의 동작에 문제가 없는 충분한 처리량(Throughput)을 갖도록 버스는 설계 된다. 이러한 구조가 확장성이 있는 일반적인 구조이기는 하지만 이렇게 하기 위해서는 SoC급의 칩을 설계 하여야 한다. 만약 IoT의 사물처럼 간단한 시스템일 때는, MCU(Micro Controller Unit)와 하드웨어 블록 1~2개가 결합된 단순한 형태로 설계하기를 원하게 된다. 본 논문은 이런 경우를 다루었다.

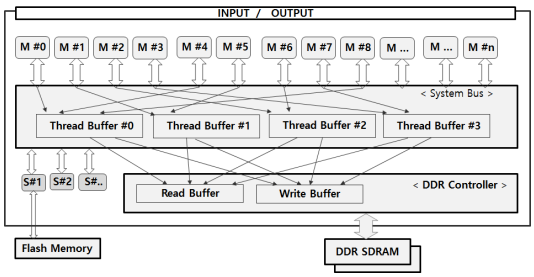

Fig. 1. Interconnection in SoC

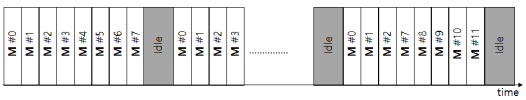

Fig. 2. Time Slot Assign, Example

2.2 IoT를 위한 간략화 된 구조

IoT 시스템은 복잡하지 않는 비교적 간단한 구성이다. 결과를 인터넷으로 보낼 때는 저속으로 동작 한다. 그러나 하위의 처리 단계에서는 고속처리가 필요하다. 특히 영상신호나 디스플레이(Display) 기능의 경우 빠

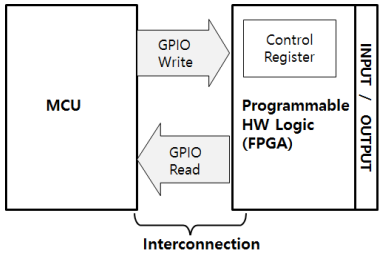

Fig. 3. Simple Control Interconnection by Using GPIO

른 하드웨어의 동작이 없으면 기능 구현이 불가능 하다. 이러한 요건을 반영한 사물의 구성은 그림 3과 같이 하였을 때 효과적으로 구현 할 수 있었다. 이와 같은 구성은 어떤 하드웨어 어떤 소프트웨어를 선택하느냐에 상관없이 구성할 수 있다. 시스템이 안정된 후, 저 전력까지 고려한다면 칩(Chip)화 까지 고려하여야 한다.

소형 시스템을 구성하는 대부분의 경우에 MCU (Micro Controller Unit)가 꼭 필요하다. 프로그램어블하게 순차적인 일 처리를 하는 MCU는 다양성이 있는 IoT에서는 꼭 필요한 요소이다. 프로그램어블한 하드웨어로직은 FPGA 이다. FPGA는 칩과 비슷한 동작 속도를 낼 수 있으면서 현장에서 쉽게 설계 할 수 있는 장점이 있다. MCU와 FPGA간의 제어통신은 GPIO(General Purpose Input Output)로 한다. 즉 GPIO를 이용해서 시스템버스의 역할을 대신 하는 것이다. 그러나 GPIO를 설정할 때는 포트(Port)간의 시간의 차이가 있다. 따라서 이렇게 하기 위해서는 상호 통신에 대한 시간 지연에 대한 대비가 되어야 한다.

IoT 시스템이라 하더라도 단순한 센스를 연결하는 것을 넘어, 카메라를 붙여서 이미지 처리를 하거나 CPU로 연산을 한 후에 클라우드에 보내야 할 때가 있다. FPGA내부에 50Mhz 정도는 클럭(clock)이 제공되기 때문에 소형의 이미지 처리는 가능하다. 아두이노 에도 C로 프로그램을 하기 때문에 많은 연산기능 처리가 가능하다. 만약 야외에서 배터리를 쓰는 상황이라 저 전력이 필요하다면, 전체 시스템을 반도체 칩화 하는 단계가 필요 하다.

2.3 MCU와 FPGA 소자의 선정 및 상호 프로토콜(Protocol)

MCU는 저렴하고 구입이 용이한 아두이노 보드를 사용한다. 이 보드는 GPIO 포트 출력을 주요 기능으로 하는 디바이스 이다. 8비트 프로세스 이고, 통합설계환경인 IDE가 개인용 컴퓨터에서 무료로 지원된다.

스케치[7]라는 화면을 통해서 C 언어로 쉽게 프로그램 할 수 있다. FPGA는 자일링스(사) 스파르탄6 버전을 사용한다. 상업용으로 사용되는 FPGA 칩이지만 이것 또한 통합설계환경인 IDE가 개인용 컴퓨터(PC)에서 무료로 지원된다. PC에서 설계된 비트 파일은 USB 다운로드를 통해서 FPGA 칩을 프로그래밍 할 수 있다. 각각의 장치는 모두 쉽게 구입이 가능하고 저렴하다. MCU와 FPGA간에는 빈번하게 통신이 이루어 져야 한다. 따라서 빠르고 간편한 프로토콜을 이용할 필요가 있다. 그림 3과 같이 구성된 경우 데이터를 주는 쪽과 받는 쪽이 있다. 데이터를 주는 쪽은 프로토콜의 약속을 지키면서 데이터를 보내는 쪽이기 때문에 중요한 이슈는 주로 데이터를 받는 쪽에 있다.

그림 4는 MCU와 FPGA간의 통신방법을 나타낸 것이다. 그림 4 의 프로토콜을 사용하여 그림 3에 있는 컨트롤 레지스터에 값을 전달하려는 것이다. 아두이노 보드는 핀 숫자가 많지 않기 때문에 Address/Data 신호를 사용해서 이 신호가 "L"이면 주소를 나타내고, "H"이면 데이터를 보내주는 형태로 하여 핀 수의 소비를 줄였다. Enable신호는 데이터가 유효한 타이밍(Timing)구간을 나타낸 것이다. 이 외에도 하드웨어와 소프트웨어 간 서로 정보를 전달하는 방법은 여러 가지가 있다. 요구하는 통신 속도, 입출력 핀 수 등 당면한 상황에 따라 선택할 수 있다. 또한 아두이노 보드 등에는 UART, SPI, I2C등의 통신 장치가 있다. 이러한 통신 장치를 이용해서 통신할 수도 있다. 다만 이러한 장치는 직렬통신이고 통신을 위한 버퍼(Buffer)를 통과해서 데이터가 나옴에 따라 상대적으로 속도가 느리다. 또한 받는 쪽도 이런 규격을 지원하는 기능을 넣어야 한고, 버퍼를 통해서 받기 때문에 즉시성이 떨어지는 면도 있다. 그리고 소프트웨어만으로도 기능을 설계 할 수도 있으나, 13개의 GPIO의 숫자 제약도 있고, 핀이 모두 개별 포트(Port)이기 때문에 포트간의 시간의 차가 있다. 1번 포트와 2번 포트를 설정하는 C프로그램 사이에 많은 프로그램 라인이 있다면 이 라인 개수에 비례해서 시간 지연이 증가 된다. 동작속도나 시간지연 등이 응용에 따라 문제 될 수도 혹은 안 될 수도 있기 때문에 이런 면을 고려해서 방법을 정하면 된다.

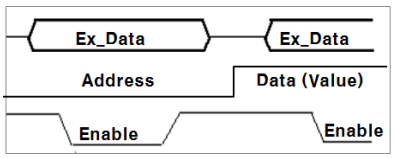

Fig. 4. Protocol Between MCU and FPGA

그림 4의 프로토콜을 이용해서 수신하려는 데이터는 그림 5에 나타난 것과 같은 컨터롤 레지스터 값이다. 컨터롤 레지스터는 주소를 가진 플립플롭(Flip-Flop) 어레이(Array)이다. 플립플롭의 Q출력으로 하드웨어를 제어하기 때문에 컨터롤 레지스터의 값이 바뀌면 바로 하드웨어의 동작이 변경된다. 그림 6은 MCU에서부터 오는 데이터를 수신하고 레지스터 테이블을 만드는 Verilog 코드 이다. 그림 6의 6라인에 "ctl_reg"는 2D 어레이로 선언되어 있다. 그림 5와 같이 16개 레지스터가 4비트 길이로 구성되어 있다. 2D 어레이 이기 때문에 ctl_reg[0]이라는 것도 1비트가 아니라 4비트로 구성되어 있다. 그림 7의 시뮬레이션을 통해서도 알 수 있다. 라인 24 이상은 같은 형태로 반복되기 때문에 표시가 생략되었다. 컨터롤 레지스터를 통해서 하드웨어를 제어하는 방법은 SoC에서는 많이 사용되는 보편적인 방법이다. 전원이 온(On)

	d3	d2	d1	d0
ctl_reg[0]	*	*	*	*
ctl_reg[1]	*	*	*	*
ctl_reg[2]	*	*	*	*
ctl_reg[3]	*	*	*	*
ctl_reg[4]	*	*	*	*
ctl_reg[5]	*	*	*	*
ctl_reg[6]	*	*	*	*
ctl_reg[7]	*	*	*	*

Fig. 5. Control Register Table for FPGA

```
1  module control_block (resetn, osc, enable, addr_val, ex_data,
2      ctl_reg[0], ctl_reg[1],ctl_reg[2],ctl_reg[3],ctl_reg[4],ctl_reg[5],
3      ctl_reg[6],ctl_reg[7],ctl_reg[8],ctl_reg[9],ctl_reg[10],ctl_reg[11],
4      ctl_reg[12],ctl_reg[13], ctl_reg[14],ctl_reg[15] );
5    input resetn, osc, enable, addr_val ;  input [3:0] ex_data ;
6    output reg [3:0] ctl_reg [15:0] ;
7    reg  [3:0] addr, data ;
8   always @(posedge osc )
9  begin
10   if (!resetn) begin addr <= 0 ; data <= 0; end
11   else begin
12      if    (enable == 0 && addr_val == 0) begin addr <= ex_data ; end
13      else if (enable == 0 && addr_val == 1) begin data <= ex_data ; end
14      else begin addr <= addr;  data <= data ; end
15   end
16  end
17  always @(posedge osc)
18 begin
19   if (!resetn) begin addr <= 0 ; data <= 0; end
20   else begin
21        case ({addr,data})
22         {4'd0, 4'd0}  : ctl_reg[0] <= 4'd0 ;
23         {4'd0, 4'd1}  : ctl_reg[0] <= 4'd1 ;
```

Fig. 6. Data Receiving Example

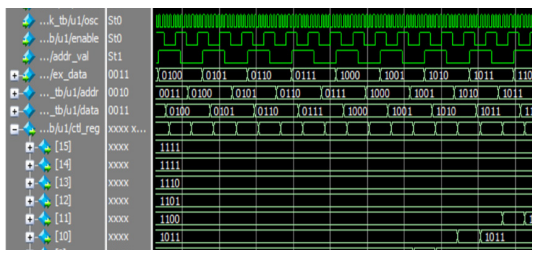

Fig. 7. Data Receiving Code, Simulation Result

된 후에 CPU에서 컨터롤 레지스터에 값을 채워 넣는 일은 가장 먼저 실행된다.

ex_data를 통해서 들어오는 값을 주소와 데이터로 분리해서 2D 레지스터 어레이에 담는다. 레지스터는 하드웨어를 제어를 하기 위한 명령어뿐만이 아니라, 초기 하드웨어 설정 파라미터(Parameter)등을 담아두는 용도로도 사용된다. 초기 설정 값은 한번 정해지면 고정될 수도 있지만, 동작 상황에 따라 동적으로 계속적인 변경을 하는 경우도 있다. 이러한 동작은 SoC에서 시스템버스를 통해서 이루어지는 기본적인 동작과 같다. 규모가 작은 시스템은 이런 구성으로 충분히 하드웨어와 소프트웨어간의 유기적인 동작을 할 수 있다.

다음은 하드웨어에서 보내주는 신호를 MCU가 받는 과정을 살펴보도록 한다. MCU는 포트를 통해서 데이터를 읽어 들이거나 데이터를 출력한다. 한 개의 핀이 한 개의 포트이기 때문에 GPIO를 통해서 읽거나 쓰는 동작을 할 때는 지연시간이 소요된다. 그림 8은 아두이노 보드에서 포트간의 지연을 측정한 것이다. 최대한 빠른 출력을 낼 때에도 4.4us의 지연이 발생하는 것을 볼 수 있다. 여러 개의 포트를 제어 하면 그만큼 비례해서 지연이 늘어난다.

그렇기 때문에 데이터를 수신 할 때에는 타이밍적인 고려가 필요 하다. 그리고 MCU가 데이터를 수신할 때는 동일 데이터는 1번만 받아야 한다. 이런 경우를 고려하여 MCU쪽에서 데이터를 수신하는 방법은 다음과 같이 하였다. 그림 9와 같이 EN신호의 “L”구간에 맞추어 데이터가 들어오도록 약속되어 있다. 즉 EN신호의 “L” 구간은 데이터가 안정이 되어 있다. 그

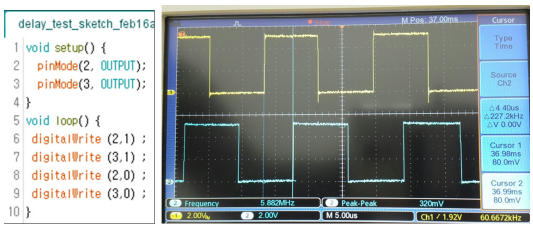

Fig. 8. Timing Delay Between Ports

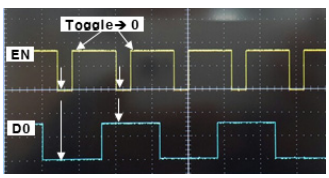

Fig. 9. MCU Input Data

```
4  void setup() {
5    lcd.begin(16, 2); // LCD 크기 설정
6    Serial.begin(115200);
7    pinMode(2, INPUT);  //  D0  (From  P0  )
8    pinMode(3, INPUT);  //  D1  (Frome P1  )
9    pinMode(5, INPUT);  //  EN  (Enable)
10   lcd.setCursor(0, 0);   }
11 int i = 0 ;
12 int Data0 ;
13 int Data1 ;
14 int Enable ;
15 int Toggle = 0 ;
16 void loop() {
17  Enable = digitalRead(5) ;
18  if (Enable == 0 && Toggle == 0 )
19     {
20      if ( i == 0 ) { lcd.clear(); lcd.setCursor(0, 0); }
21      else if ( i == 16 )  { lcd.setCursor(0, 1) ;  }
22      else if ( i == 32 )  { i = 0 ; lcd.clear(); lcd.setCursor(0, 0); }
23      else  {  }
24     Data0 = digitalRead (2) ;  // D2 Read as D0
25     Data1 = digitalRead (3) ;  // D3 Read ad D1
26     Toggle = 1 ;
27      if      (Data1 == 0 && Data0 == 0) { Serial.println("0"); lcd.print("0") ; }
28      else if (Data1 == 0 && Data0 == 1) { Serial.println("1"); lcd.print("1") ; }
29      else if (Data1 == 1 && Data0 == 0) { Serial.println("2"); lcd.print("2") ; }
30      else if (Data1 == 1 && Data0 == 1) { Serial.println("3"); lcd.print("3") ; }
31      else { }
32      i = i + 1 ;
33      }
34    else if ( Enable == 1 && Toggle == 1 ) { Toggle = 0 ; }
35    else { }
36      }
```

Fig. 10. MCU Side, Data Receiving & LCD Display

림 10은 이 데이터를 수신하고, 또한 LCD로 결과를 표시하는 코드 이다. 아두이노의 스케치는 setup()함수와 loop()함수를 가지고 있다. setup()함수의 { } 내에는 초기 설정을 하는 코드를 넣고, loop()함수의 { } 내에는 반복적으로 실행할 코드를 넣는다. “Enable== 0 && Toggle==0”일 때가 안정된 타이밍 구간이기 때문에 이때 데이터를 읽어 들인다. 그림 10의 24~26 라인(Line)에서 digitalRead() 함수를 이용해서 데이터를 읽고 난 뒤 바로 “Toggle = 1”로 바꾸어 동일 데이터가 2번 이상 읽어지는 것을 방지 하였다. 명령어는 순서대로 실행되기 때문에 digitalRead() 바로 다음에 “Toggle = 1”로 바꾸어 주어야 한다. EN 신호가 “H”가 되면 “Toggle = 0”으로 하여 다시 데이터를 받을 수 있도록 하였다. 이 상태에서 “Enable== 0”이 되면 다시 데이터를 읽을 수 있게 되는 것이다. 그림 10의 27~30라인은 결과 데이터를 LCD로 표시하는 코드 이다. 서로 정확하게 데이터를 주고받는 방법은 더 많다. 데이터가 중복되거나 빠지지 않도록 하는 방법이면 되고, 단지 하드웨어와의 통신 속도는 빠른 것이 유리 하다.

2.4 FPGA 구현설계

하드웨어 로직을 설계하고, FPGA에 다운로드 하

여 시스템을 구성하면 많은 하드웨어를 쉽게 설계 할 수 있을 뿐만 아니라 안정적인 시스템을 구축할 수 있다. FPGA 구현설계라는 것은 Verilog 언어를 사용해서 기능설계를 하고

시뮬레이션(Simulation)을 통해서 검증이 완료된 Verilog 코드를 FPGA로 구현하는 것을 말한다. 다양한 경우를 다루어야 하는 IoT의 응용에서는 매우 유용한 방법이 된다.

FPGA 구현이라는 것을 하기 위해서는 매우 구체적으로 지정하여야 한다. 그러나 다른 FPGA도 유사한 형태로 되어 있기 때문에 다른 곳에서도 유사하게 응용 할 수 있다. FPGA는 Xilinx(사)의 SPARTAN-6 계열을 사용하였고, 품명은 XC6SLX9이고, 패키지(Package)는 TQG144이다. 그림 11은 개발 모듈형태로 만들어진 FPGA 보드 이다. 이것은 보드 개발의 노력을 대폭 줄일 수 있게 해준다. 주변 부품의 실장이 이미 되어 있기 때문에 전원을 인가하고 입력과 출력을 연결하여 바로 사용할 수 있다. 설계된 하드웨어의 비트파일을 USB로 다운로드하면 바로 동작 실행이 가능하다.

FPGA 구현설계의 순서는 그림 13[8]와 같이 논리합성(Synthesize-XST), 구현설계(Implementation Design), 프로그래밍 파일 생성(Generate Programming File), 목표하는 소자의 설정(Configure Target Device)의 단계를 거쳐서 비로소 동작하는 FPGA를 만든다. 첫 단계로, 시뮬레이션으로 논리 검증 완료된 Verilog 코드가, 실제 합성이 가능한 코드인가를 체크 한다. 논리 시뮬레이션에서 통과 되었다 하더라도 FPGA로 구현이 될 수 없는 코드이면 오류가 난다. 논리가 맞다 고해서 반드시 로직 구현이 가

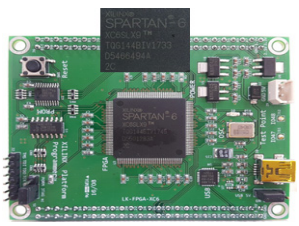

Fig. 11. FPGA Development Board

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan6
Device	XC6SLX9
Package	TQG144
Speed	-2

Fig. 12. FPGA Device Selection

Fig. 13. FPGA Implementation Sequence

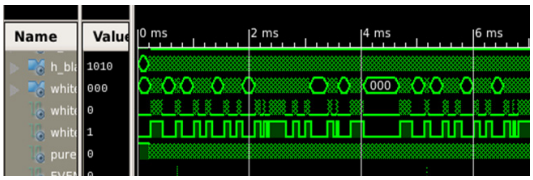

Fig. 14. Simulation after Synthesize

능한 것은 아니기 때문이다. 이상이 없으면 논리 합성(Synthesize) 하여 로직 회로도(Net List)를 만든다. 논리합성을 한 뒤는 시뮬레이션 해서 동작에 문제가 없는지 그림 14과 같이 기능과 타이밍을 확인한다.

두 번째 단계는 구현설계 단계이다. FPGA 칩 내에서 로직 게이트(Gate)를 배치하고 실제 게이트 간을 연결하는 행위를 한다. 또한 사용자용 입출력 핀에 대해서는 FPGA에서 핀의 배치도 하여야 한다. 그림 15는 .ucf 파일을 통해 핀의 위치를 설정하는 것을 보인 것이다. 처음 시도에서는 핀의 위치를 지정하지 않고 구현설계를 진행한다. 그리고 난 뒤 "Pinout Report "를 통해서 사용자 입출력이 어느 핀에 할당되었는지를 확인 한다. 즉 핀 지정을 툴의 자동 배치에 맡기는 것이다. 만약 회로의 규모가 크면서 또한 사용자 입출력 핀 위치도 강제로 지정하게 되면, 툴이 타이밍을 맞추지 못하여 구현설계 단계를 실패하게 된다. 따라서 처음에는 툴의 자동배치에 맡기는 게 유리하다. 회로를 설계하여 실험을 하다보면 회로 수정사항이 발생하게 되고, 다시 구현설계를 해야 하는 경우가 발생한다. 이때는 그림 15에 있는 것과 같은 .ucf 파일을 통해서 핀의 위치를 고정하여 지정하여야 한다. 그렇지 않으면 실험하는 보드의 입출력을 계속 수정하여야 하기 때문이다. 한다. 또한 1차에 비해 미세한 회로 수정만 있었기 때문에 IDE 툴이 타이밍을 못 맞출 위험은 매우 낮다.

세 번째 단계는 프로그래밍 할 파일을 생성하는 단

```
1 NET "osc"          LOC = P50 | IOSTANDARD = "LVCMOS33";
2 NET "resetn"       LOC = P46 | IOSTANDARD = "LVCMOS33";
3 NET "enable"       LOC = P45 | IOSTANDARD = "LVCMOS33";
4 NET "addr_val"     LOC = P21 | IOSTANDARD = "LVCMOS33";
5 NET "ex_data[0]"   LOC = P47 | IOSTANDARD = "LVCMOS33";
```

Fig. 15. FPGA Pin Assignment

계이다. 이때는 IDE툴이 .bit 파일이라는 FPGA에 다운로드할 파일을 만드는 동작을 한다. 네 번째는 실험보드에 있는 FPGA에 .bit 파일은 다운로드하는 마지막 단계이다. 그림 16은 자일링스 USB 다운로드를 통해서 FPGA에 .bit 파일을 전송하여 실험하는 상황을 보인 것이다. 실험초기는 bit 파일을 FPGA내부에 전송하여 빠르게 반복 실험을 진행하고, 회로가 안정되면 FPGA 보드 내에 있는 플래쉬메모리(Flash Memory)에 bit 파일을 저장하여 단순화된 보드의 형태로 사용한다.

이렇게 생성된 데이터는 IoT용 허브(Hub)를 통해서 인터넷과 접속된다. IoT 허브 장치는 통신기능을 포함하고 있다. 인터넷 클라우드에서 데이터를 판단하고 액션(Action)하는 방법은 “ARTIK 클라우드를 통한 분산된 원격 IoT 센서 데이터 모니터링 및 제어”[1]를 통하여 발표하였기 때문에 여기서는 생략하도록 한다. 아두이노 보드 대신에 더 상위 기능의 라즈베리파이를 사용하면 동일한 구성으로 더 좋은 성능을 낼 수 있다.

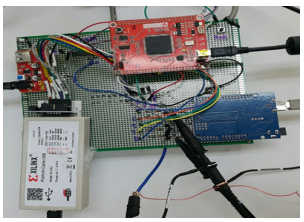

Fig. 16. Finial .bit File Downloading & Development

Ⅲ. 결 론

IoT 시스템의 사물 부분을 효율적으로 설계하기 위한 방법을 제시 하였다. IoT 시스템은 다양성을 필요로 한다. 본 튜토리얼 논문에서는 많은 부류의 사람들이 쉽게 구입할 수 있는 부품과 개인용 PC에서도 쉽게 구동되는 IDE 툴을 활용하였다. 회로 규모가 작고 비교적 느린 동작이 허용 되는 IoT의 특성을 고려하여, 허용이 가능한 부분은 성능 양보를 하여 줄이고, 대신 많은 자유를 필요로 하는 부분은 유연성을 살렸다. IoT의 설계는 여러 계층을 설계하여야 하기 때문에 하드웨어와 소프트웨어를 동시에 설계하여야 한다. 하드웨어설계의 자유도를 가질 수 있도록 FPGA(Field Programmable Gate Array)를 사용하였고, 다양성과 직관적인 프로그래밍(Programming)이 가능한 아두이노(Arduino) 보드를 사용하여 소프트웨어를 설계 하였다. 두 가지가 결합되어 동작할 수 있도록 통신하는 방법도 정하였다.

IoT는 소규모 이고, 요구되는 성능 지표 수준이 상대적으로 낮다. 그렇기 때문에 SoC로 구현하였다 하더라도 저 전력이외는 강점을 찾기가 어렵다. 따라서 대량 양산 이전까지는 제안된 방법이 효과적이다. 이 방법은 시간, 투자, 노력, 사용 부품 등의 비용을 최소로 하면서도 높은 설계 자유도를 가질 수 있도록 한다.

References

[1] J.-H. Bae and J.-T. Kim, “Distributed remote iot sensor data monitoring and control through the ARTIK clou,” *J. KICS*, vol. 43, no. 12, pp. 2100-2106, Dec. 2018.

[2] Xilinx home page SPARTAN6, https://www.xilinx.com/products/silicon-devices/fpga/spartan-6.html

[3] Arduino home page Aduino UNO, https://www.arduino.cc/en/Guide/ArduinoUno

[4] A. Saxena, M. Tyagi, and P. Singh, “Digital outing system using RFID and raspberry Pi with MQTT protocol,” *2018 3rd IoT-SIU*, pp. 1-4, 2018.

[5] A. Saxena, M. Tyagi, and P. Singh, “Study growth environmental monitoring and controlling platform for hydroponic,” *J. KICS,* vol. 41, no. 09, pp. 1132-1140, Sep. 2016.

[6] E.-M. Ahn and D.-H. Kim, “Implementation of integrated platform of face recognition CCTV and home IOT,” *J. Digital Contents Soc.*, vol. 19, no. 02, pp. 393-399, Feb. 2018.

[7] Arduino home page Sketch, https://www.arduino.cc/en/tutorial/sketch

[8] Xilinx home page Download, https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools/14_7-windows.html

배 점 한 (Jum-Han Bae)

1985년 : 경북대학교 전자공학과 졸업
2013년~현재 : 성균관 대학교 전자전기컴퓨터 공학 박사과정
1985년~2014년 : 삼성전자 DMC 연구소
2015년~2016년 : SK하이닉스 메모리시스템 연구소
현재 : 숭실대학교 전자정보공학부 교육형 산학중점 교수
<관심분야> SoC Design, Embedded System, IoT.
[ORCID:0000-0002-4865-3356]

김 종 태 (Jong Tae Kim)
한국통신학회 논문지 제 34권 5호 참조
현재 성균관대학교 전자전기공학부 교수
[ORCID:0000-0003-0290-0865]