Simulation and Measurement of an Internet of Things Implementation of a Programmable Digital Inductor

Tyler C. Major, Kona Pranay Shekhar, James M. Conrad, and Thomas P. Weldon
Department of Electrical and Computer Engineering
University of North Carolina at Charlotte
Charlotte, NC, USA
tmajor1@uncc.edu, pkona@uncc.edu, jmconrad@uncc.edu

***Abstract*—The theory, design, and implementation of a programmable Internet of Things digital inductor circuit is presented. Since the inductor is implemented in digital signal processing, the values for the inductance are easily modifiable through an Internet of Things interface, enabling remote access to the physical hardware. The system was implemented on a commercially-available microntroller board using a client-server scheme over ethernet connection. A control interface running on a laptop was then used to change the inductance remotely. Simulation and implementation of the design are presented and are shown to match at a sample rate of 100 kilosamples per second. Since the design method constitutes a digital approach, performance is expected to scale with higher-speed hardware.**

***Index Terms*—Digital Inductor, Embedded Systems, Internet of Things.**

I. Introduction

The ubiquitous presence of the internet combined with the availability of small inexpensive microcontrollers is contributing to the development of Internet of Things (IoT) and associated applications [1]. Indeed, researchers are only beginning to develop new systems that take advantage of this emerging confluence of high connectivity and distributed computing power. Considerable opportunity exists at both the network level, in taking advantage of coordinated utilization of many devices in the IoT, and at the device level, in creating new "Things." The present investigation focuses on the device level, in particular an IoT-controlled tunable digital inductor.

To demonstrate the proposed IoT-controlled digital inductor, a client-server internet model was chosen. In this, a laptop was used as a server and provided the user interface to send commands to the remote microcontroller. Client software runs on the microcontroller to communicate with the laptop server, and update parameters such as the desired inductance or resistance. In addition, the microcontroller ran the signal processing necessary to implement inductance, and included on-board analog-to-digital converter and digital-to-analog converter. The resulting system provides a remotely-tunable digital inductance that can be useful in a range of applications such as impedance matching networks [2]. As is well known, stability depends upon parasitic resistance in addition to capacitance and inductance [3]. Due to this, the paper adds in a resistance to the purely inductive system to induce stability for the closed system for demonstration purposes. This design is re-programmable through an Internet of Things scheme to allow for on-the-fly reprogramming of the system.

The following section first describes the overall IoT approach and system architecture. The subsequent section summarizes theory of the digital inductor circuit and provides simulation results. The final section describes measured results for a prototype, in good agreement with simulation results.

II. IoT System Architecture and Design

The system in Fig. 1 was implemented as an Internet of Things interface concept. The micro controller board was an NXP FRDM-K64F microcontroller board. The FRDM-K64F board houses a 120Mhz ARM Cortex-M4 Core with Floating point unit and DSP, along with a 16-bit ADC, 12-bit DAC, and Ethernet interface. The Ethernet socket interface uses TCP protocol for communication, so the client can be accessed from anywhere as long as the client and server are on the same network, while using a gateway could provide accessibility across the Internet, i.e resolve the hostname. For purposes of demonstration, it is not necessary to hard-code an IP address, since both the server-client are on the same network connected point-to-point via an Ethernet cable.

In order to implement the digital inductor on the FRDM-K64F board, the on-board ADC and DAC are used. The voltage input to the ADC in Fig. 1 digitizes the analog input voltage $v_{in}(t)$ to form the discrete time output $v_{in}[n]$. The DAC output $v_{dac}[n]$ is given through $V_{dac}(z) = V_{in}(z)H(z)$,

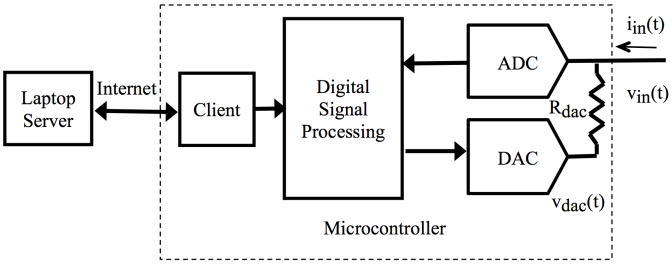

Fig. 1. Proposed IoT-controlled tunable digital inductor. A laptop server interfaces with a client running on the microcontroller, and passes design parameters to the digital signal processing software on the microcontroller. The on-board ADC and DAC on the microcontroller are connected to resistor R_{dac}, and connect to the analog port with voltage $v_{in}(t)$ and current $i_{in}(t)$. Signal processing is used to establish an inductive impedance at the port $v_{in}(t)$.

```
//Server IP Address
const char* SERVER_ADDRESS = "10.42.0.1";
//Server Port Number
const int SERVER_PORT = 9009;
```

Fig. 2. Code snippet of the client - side which shows the IP address and Port number of the server.

where $H(z)$ is the z-transform of the impulse response $h[n]$ characterizing the digital signal processing.

The voltage output DAC creates continuous time output voltage $v_{dac}(t)$. Finally the continuous time input current $i_{in}(t)$ is then $(v_{in}(t) - v_{dac}(t))/R_{dac}$. The z-transform $H(z)$ determines the behavior of the overall circuit. The architecture of the signal processing follows along lines similar to [4], except that a series resistance component is added to help prevent instability and a voltage-output DAC is used in conjunction with R_{dac}, as shown in Fig. 1.

Using Python socket interfaces [5], basic socket calls are implemented to connect and establish socket connection on the server side. The FRDM-K64F supports the MBED library, which provides the Ethernet Interface library support, although discontinued now, the routines and the library is still compiled by the MBED online IDE. Hence, the client-side socket calls are C++ code. Use of DHCP protocol enables the use of dynamic IPs, rather than defining static IP for either the Master or Slave. One care to be taken, is that the client code has the correct IP address and port number of the server, and this is defined in the client by the code snippet shown in Fig. 2.

The server-side script has to be executed on the computer/laptop system before the client program runs, because the server-side program waits for the client to connect and only after a successful connection displays the interface for the user to enter data. As shown in the blue box of Fig 3, the server waits for the client to be connected after the script starts. Once a client is successfully connected, the client's IP address is displayed.

A computer system or laptop with RJ-45 port, Ethernet support, and Python support (v 2.7.10) is required to interface with the Ethernet port of FRDM-K64F. A server-client architecture is implemented, where the computer system executing a python based script is the server and the FRDM-K64F is the client which receives the design parameters from the server. The python script provides an interface for the user to input the values of series resistance R_{ser} and inductance L. The

```
python Server_socket.py
Server started on port 9009
Client (10.42.0.107, 49153) connected

Follow the Function code (FC) based protocol for changing data
01 - Change Resistance value in ohms, e.g 01 200
02 - Change Inductance value in mH, e.g 02 20
03 - Send data
```

Fig. 3. Execution of the server-side script. The blue box indicates the socket call on the server side that waits for the client to connect and displays the client's IP address on successful communication. The blue box displays the FC based options for the user to enter the data.

```
$ python Server_socket.py
Server started on port 9009
Client (10.42.0.107, 49153) connected

Follow the Function code (FC) based protocol for changing data
01 - Change Resistance value in ohms, e.g 01 200
02 - Change Inductance value in mH, e.g 02 20
03 - Send data
[R = 100.0 ohms, L = 20.0 mH] 03
Values to be sent: Inductance = 20.0 mH, Resistance = 100.0 ohms.
System Stable : 0.09375
Values sent successfully
[R = 100.0 ohms, L = 20.0 mH] 01 70
[R = 70.0 ohms, L = 20.0 mH] 02 40
[R = 70.0 ohms, L = 40.0 mH] 03
Values to be sent: Inductance = 40.0 mH, Resistance = 70.0 ohms.
System Stable : 0.0375
Values sent successfully
```

Fig. 4. Execution example of the system. The example shows how to change the values and send the data to the client.

system also makes use of Function Code (FC) based system for updating values of R_{ser} and L. The FCs for the system are given in Table I. FC 01 is used for modifying value of Resistance, FC 02 for inductance, and FC 03 to send design parameters to the client (FRDM-K64F).

TABLE I
FUNCTION CODES

Function Code	Usage Example	Description
01	01 100	Modify the resistance value to 100 Ω
02	02 40	Modify the Inductance value to 0.04 H
03	03	Send data to the FRDM-K64F

Once the connection is established between the server and client, the Function Code list is displayed to the user along with the usage example as shown by the yellow box of Fig 3.

The current values of R_{ser} and L are displayed to the user as part of the user prompt, so that the user can know the current value of both the parameters on the server. These values might not be the same values as in the client. As a security measure the values sent by the server are not directly implemented by the client, FRDM-K64F. FRDM-K64F has a programmable Switch interrupt, which is used to accept the new values from the server. Once the values are sent from the server, the values will be stored in a buffer on the FRDM-K64F. For the purpose of demonstration, the difference equation in the system will be updated only when switch 2 is pressed and the new values accepted.

The server side calculates the stability as discussed in [3], and displays this before sending the data. There is no restriction on transmitting unstable values, as the stability analysis is only with respect to a $50\,\Omega$ system, and hence the overall circuit can be stable for other source impedance values. Fig. 4 shows the execution of different sets of values and transmitting them to the board.

III. Digital Inductor Design

The design of the digital inductor follows along lines similar to [4], except that a series resistance component is added to help prevent instability and a voltage-output DAC is used in conjunction with R_{dac}, effectively a Thèvenin equivalent

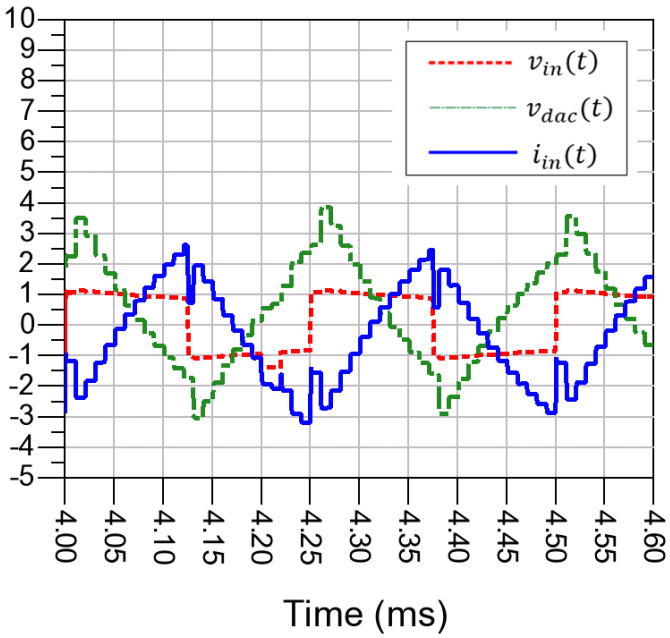

Fig. 5. ADS time-domain simulation using a $50\,\Omega$ 4 KHz square wave input signal at $T = 10\,\mu$s, for $L = 0.02\,$H, $R_{ser} = 100\,\Omega$, and $R_{dac} = 1000\,\Omega$. The red dashed line indicates the input voltage to the ADC $v_{in}(t)$. Thin green dot-dashed trace is the output on the DAC, v_{dac}. Blue trace is the current $i_{in}(t)$ in mA.

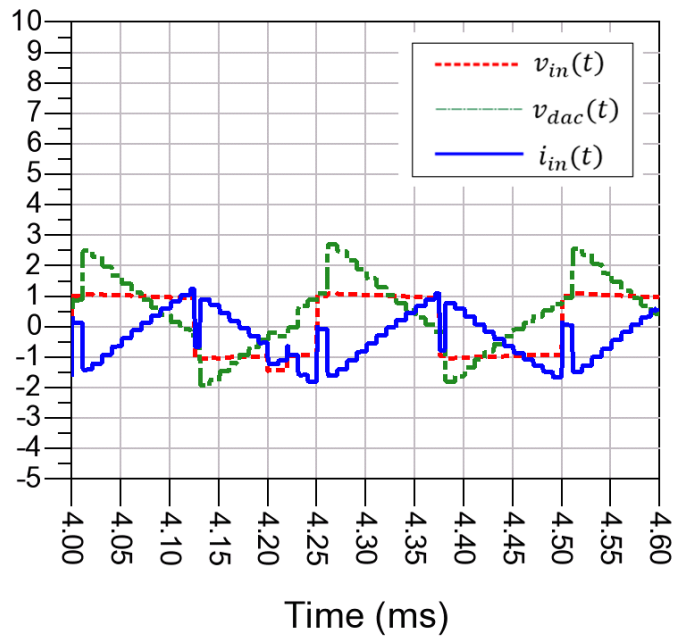

Fig. 6. ADS time-domain simulation using a $50\,\Omega$ 4 KHz square wave input signal at $T = 10\,\mu$s, for $L = 0.04\,$H, $R_{ser} = 100\,\Omega$, and $R_{dac} = 1000\,\Omega$. The red dashed line indicates the input voltage to the ADC $v_{in}(t)$. Thin green dot-dashed trace is the output on the DAC, v_{dac}. Blue trace is the current $i_{in}(t)$ in mA.

of [4]. In this approach, the z-transform $H(z)$ and associated difference equation determines the behavior of the overall circuit. The following paragraphs develop the particular case where a digital inductor is considered.

To begin the development of the inductor design, first consider a voltage across an inductor $v(t) = Ldi(t)/dt$, which has a simple discrete-time approximation of the derivative being $v(t) \approx L(i[n] - i[n-1])/T$, where T is the sampling period. Approximating the input current as $i_{in}(t) \approx [v_{in}(t) - v_{dac}(t)]/R_{dac}$ and substituting into the equations yields the difference equation

$$v_{in}(t) \approx \frac{L}{R_{dac}} \frac{(v_{in}[n] - v_{in}[n-1]) - (v_{dac}[n] - v_{dac}[n-1])}{T}. \tag{1}$$

Solving for $v_{dac}[n]$ gives

$$v_{dac}[n] - v_{dac}[n-1] = v_{in}[n](1 - \frac{R_{dac}T}{L}) - v_{in}[n-1]. \tag{2}$$

To help improve control of the stability of the digital inductor, a series resistance R_{ser} is also added to the signal processing. Adding this series resistor gives $v(t) = Ldi(t)/dt + i(t)R_{ser}$. Solving, as before, for $v_{dac}[n]$ then yields

$$v_{dac}[n] = \frac{v_{in}[n](1 + \frac{R_{ser}T - R_{dac}T}{L}) - v_{in}[n-1] + v_{dac}[n-1]}{(1 + \frac{R_{ser}T}{L})}. \tag{3}$$

IV. Simulation

The signal processing of Fig. 1 with the laptop server portion being the implementation from Fig. 3 was simulated in ADS for 2 sets of values, an inductance of $0.02\,$H and a series resistance of $100\,\Omega$, and an inductance of $0.04\,$H and a series resistance of $100\,\Omega$. A clock frequency of 100 KHz was used, corresponding to $T = 10\,\mu$s, and driven by an input source of impedance $50\,\Omega$. The simulation was done using ideal switches and ideal amplifiers as in [4], where the gains of the amplifiers establish the difference equation coefficients in (3). In addition, a latency of 0.9 μs was added to account for ADC conversion time and computational latency. The results of the simulations are shown in Fig. 5 and Fig. 6, with the blue trace representing the current $i_{in}(t)$.The dashed red line is the input voltage to the ADC $v_{in}(t)$, the thin dot-dashed line is the voltage output of the DAC $v_{dac}(t)$. Note that, the voltage $v_{in}(t)$ corresponds to the slope of the current $di_{in}(t)/dt$, as expected for an inductor, where $v = Ldi/dt$.

V. Experiment and Measurements

The IoT digital inductor of Fig. 1, with Fig. 3 acting as the laptop server, was implemented using the NXP FRDM-K64F microcontroller board. It was measured using a 4 KHz square wave input signal, with an amplitude of 0.2 V and an offset of 0.750 V at 50% duty cycle. The oscilloscope is configured to display the waveform of the ADC input $v_{in}(t)$, the DAC output $v_{dac}(t)$, and the input current $i_{in}(t)$ in mA (current in mA computed by subtracting the voltages across $R_{dac} = 1000$).

Fig. 7 shows measured results for the prototype with a digital inductor having parameter values of $L = 0.02$ H, $R_{ser} = 100\,\Omega$, $T = 10\,\mu$s, $R_{dac} = 1000\,\Omega$, and with a $50\,\Omega$ source driving the circuit. The red trace is the voltage input

to the ADC, $v_{in}(t)$. The green trace is the voltage output of the DAC $v_{dac}(t)$. The blue trace is the input current $i_{in}(t)$ in mA. When the slope of the current $i_{in}(t)$ is positive, the input voltage $v_{in}(t)$ is positive, and when the slope of the current $i_{in}(t)$ is negative, the input voltage $v_{in}(t)$ is negative, as expected for a positive inductor. For the observed peak input voltage of $\approx$ 25 mV, the observed slope of current is 110μA/100μs = 1.1 A/V, so with $v = Ldi/dt$, $L = 0.023$ H.

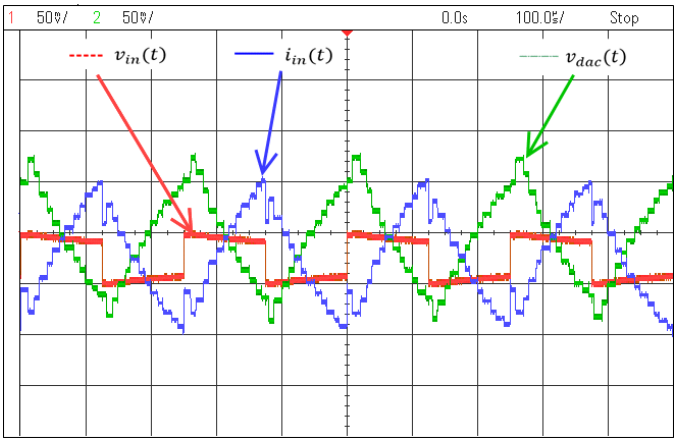

Fig. 7. Measured data at $T = 10\,\mu$s, for $L = 0.02$ H, $R_{ser} = 100\,\Omega$, $R_{dac} = 1000\,\Omega$, with 4 KHz square wave input $v_{in}(t)$, plotted at 100 μs and 50 mV per division, and a $50\,\Omega$ source driving the circuit. Red trace indicates the input to the ADC $v_{in}(t)$ from the function generator . Green trace is the output voltage of the DAC, $v_{dac}(t)$. Blue trace is the current $i_{in}(t)$. For the observed peak input voltage of $\approx$ 25 mV, the observed slope of current is 110μA/100μs = 1.1 A/V, so with $v = Ldi/dt$, $L = 0.023$ H.

Fig. 8 shows measured results for the prototype with a digital inductor having parameter values of $L = 0.04$ H, $R_{ser} = 100\,\Omega$, $T = 10\,\mu$s, $R_{dac} = 1000\,\Omega$, and with a $50\,\Omega$ source driving the circuit. The red trace is the voltage input to the ADC, $v_{in}(t)$. The green trace is the voltage output of the DAC $v_{dac}(t)$. The blue trace is the input current $i_{in}(t)$ in mA. For the observed peak input voltage of $\approx$ 25 mV, the observed slope of current is 50μA/100μs = 0.5 A/V, so with $v = Ldi/dt$, $L = 0.05$ H.

The complete experimental setup is shown in Fig. 9. The setup consists of a laptop with an RJ-45 LAN port and Ethernet interface for communication with FRDM-K64F. The FRDM-K64F is powered up via the on-board mini usb connected to the laptop. On power-up, the socket interface is established, using the Ethernet interface. The interface provides easy access to change the values of R_{ser} and L, without having to modify the program. Once the values are received by the micrcontroller, they are executed accordingly and the waveforms can be seen on the oscilloscope. There is no limitation or restriction on modifying the values on a single program execution. The waveforms displayed in Fig 7 and Fig 8 were executed in a single program execution, using the interface to modify the values of L. Hence by comparing the simulation waveforms in Fig 5 and Fig 6 with measured data from Fig 7 and Fig 8, we can infer that the system provides a programmable inductance.

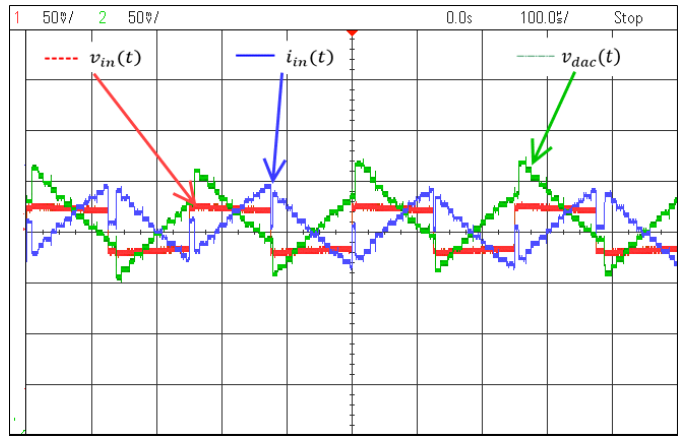

Fig. 8. Measured data at $T = 10\,\mu$s, for $L = 0.04$ H, $R_{ser} = 100\,\Omega$, $R_{dac} = 1000\,\Omega$, with 4 KHz square wave input $v_{in}(t)$, plotted at 100 μs and 50 mV per division, and a $50\,\Omega$ source driving the circuit. Red trace indicates the input to the ADC $v_{in}(t)$ from the function generator . Green trace is the output voltage of the DAC, $v_{dac}(t)$. Blue trace is the current $i_{in}(t)$. For the observed peak input voltage of $\approx$ 25 mV, the observed slope of current is 50μA/100μs = 0.5 A/V, so with $v = Ldi/dt$, $L = 0.05$ H.

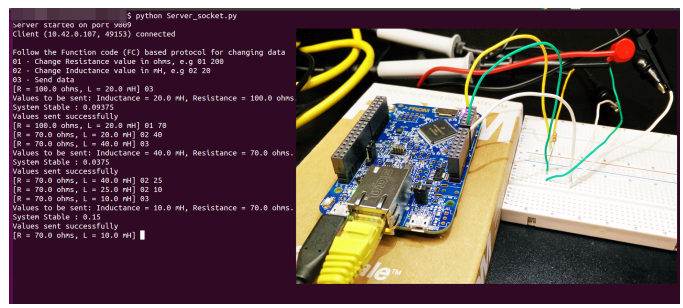

Fig. 9. Implementation setup displaying the FRDM-K64F microcontroller connected via ethernet to a laptop running the python based socket script.

VI. Conclusion

An IoT-controlled adjustable digital discrete-time inductor was presented, including design theory, simulated and measured results, and a programmable Internet of Things interface. The measured results agree with the simulated results, and the IoT interface allows remote adjustment and tuning of inductance.

References

[1] J. A. Stankovic, "Research directions for the internet of things," *Internet of Things Journal, IEEE*, vol. 1, no. 1, pp. 3–9, Feb. 2014.

[2] S. E. Sussman-Fort, "Matching network design using non-Foster impedances," *Int. J. of RF and Micro. Comp.-Aided Eng.*, vol. 16, no. 2, pp. 135–142, 2006.

[3] T. P. Weldon, J. M. C. Covington, K. Smith, and R. S. Adams, "Stability conditions for a digital discrete-time non-Foster circuit element," in *2015 IEEE Antennas and Propagation Society International Symposium (APSURSI)*, Jul. 2015.

[4] T. P. Weldon, J. M. C. Covington, K. Smith, and R. S. Adams, "Performance of digital discrete-time implementations of non-Foster circuit elements," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 2169–2172.

[5] M. Gordon, "An introduction to network programming the python way [book review]," *Distributed Systems Online, IEEE*, vol. 6, no. 10, 2005.