AN-CA-002
BeagleBell A5C

J. Haas, J. Frederickson, J. Liu, G. Umlauf
DEC 2013

Overview

This application note will outline the functionality of the BeagleBell A5C, an Internet-enabled smart doorbell. The BeagleBell was developed using Python on the BeagleBone Black as a final project for Rowan University's Computer Architecture class. The goal was to create a doorbell that would send a picture of the visitor to the homeowner's smartphone.

Background

The BeagleBone Black is relatively inexpensive and powerful development platform. It is based on the ARM architecture, which is an instruction set used by billions of electronic devices today. It is a Reduced Instruction Set Computing (RISC) architecture developed and maintained by ARM Holdings. The most recent version is ARMv8-A, which can be used for 32-bit or 64-bit applications. Among other requirements, the specification includes 31 general purpose registers. The BeagleBone Black features a 1GHz AM335x ARM Cortex-A8 TI Sitara Microprocessor, as well as a floating point unit.

The BeagleBone Black also features numerous I/O ports, including micro USB, USB, micro HDMI, micro SD, ethernet, GPIO pins, and analog pins. We used the default Ångström Linux distribution as the operating system for the BeagleBone Black. In addition, the team used Bit Bucket for version tracking and programming collaboration. The BeagleBone Black is shown below in Figure 1.

Figure 1. The BeagleBone Black.

Problem

To create a better doorbell using the BeagleBone Black. Pushing the button will send a picture of the visitor to the homeowner via text message or e-mail.

Inputs:

- Push button via GPIO
- Webcam via USB
- Config file

Outputs:

- Status via terminal
- Email via ethernet

Approach

This project used the BeagleBone Black to do all of the processing. The code was written in Python, using external libraries to easily implement input and output without worrying about low level interfacing. We used OpenCL for image capture from the webcam, Python's built-in email library for email/SMS, and Adafruit's BeagleBone I/O library for interfacing with hardware peripherals.

Our main module beaglebell.py can be found in Appendix A. It contains an infinite while loop (the program can be exited via Ctrl+C at the terminal).

Inside the loop, it stores the last state of the button, then checks the new state of the button. If the button changed from 0 (not pressed) to 1 (pressed), then we take a picture and send an e-mail. The main module also reads user configurable settings from the text file beaglebell.conf.

First, the button was attached to the GPIO headers on the BeagleBone Black using a breadboard, following a guide on Adafruit [1]. The headers are shown below in Figure 2.

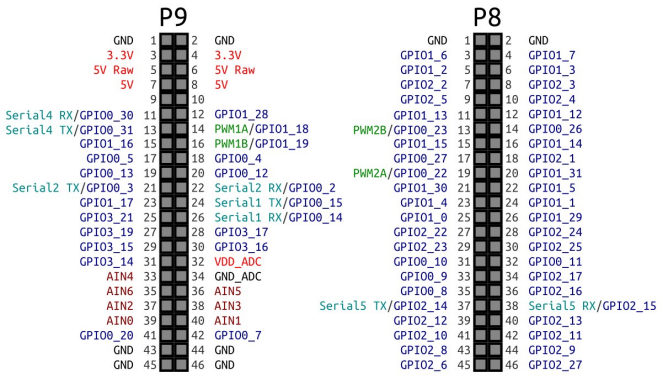

Figure 2. BeagleBone Black I/O headers.

The input pin, P8.12 “GPIO1_12”, was connected across a 1k resistor to P8.2 “GND”. The button was placed between the input pin and P9.3 “3.3V”. When button is open, the input sees 0V because it is connected to ground, producing a logic 0 in Python. When the button is pressed, the input sees 3.3V, producing a logic 1 in Python. The 1k resistor dissipates power to prevent a short circuit between the 0V and 3.3V pins. The schematic is shown below in Figure 3.

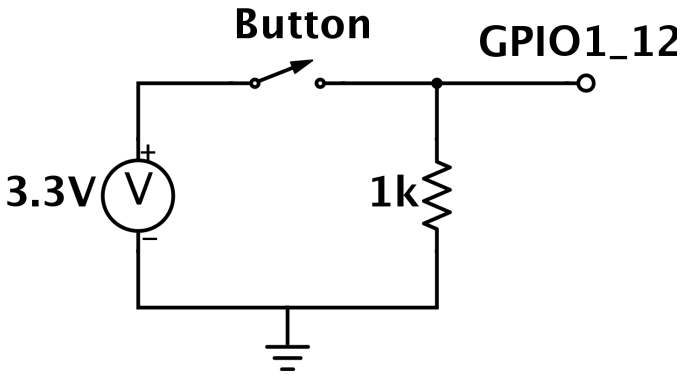

Figure 3. Schematic showing button connections.

We also made a testing module, led-button.py, that ties the state of the USR0 LED on the BeagleBone Black to the state of the button. This makes it easy to check if the button is connected correctly, and can be found in Appendix B.

Once a button press is detected, the take_picture() method in webcam.py is called, passing the path of the file to be created as a parameter.

In the webcam class (Appendix C), a new video capture object (`cv2`) is created. This functions as an interface to the camera using OpenCV. The height and width parameters are set to 320x240, and a single frame is saved to a local file.

If the image file was created successfully, the driver then passes the file path to the emailer class. Two objects are created: a MIMEMultipart object representing the email to be sent, and a MIMEImage object containing the attached image. The addresses in the “to” field are loaded from the beaglebell.conf configuration file. A connection to the chosen SMTP server is then established, and the message is sent. In the case of text message, an SMS gateway is used, which is supplied by many cell phone carriers. Basically, sending an e-mail to the SMS gateway results in a text message on the target phone.

Responsibilities

Joshua Haas
- Main driver: beaglebell.py
- Configuration: beaglebell.conf

Jonathan Frederickson
- Taking pictures: webcam.py

Graham Umlauf
- Sending mail: emailer.py

Justin Liu
- Connecting the button
- Testing the button: led-button.py

Results

The BeagleBell worked as intended, sending a picture to the addresses listed in the configuration file when the button is pressed. This was always successful when using email addresses. When

using SMS gateways, however, only Verizon and AT&T consistently worked. Sprint stripped the picture from the message and only sent the text and T-Mobile did not work at all. Other carriers were not tested.

Discussion

The email and button handling code worked as expected. However, when we attempted to take a picture with a resolution of 640x480, the image was not fully formed. After some research, we realized that the issue was in the Video4Linux utility that we were using to interface with the camera. It turns out that the decoder Video4Linux was using to decompress the JPEG-encoded images from the camera uses a lot of floating point, and the BeagleBone's FPU just isn't fast enough. Another BeagleBone user had written code to capture uncompressed images instead, but doing so is very webcam specific [2]. Due to time constraints, we did not attempt to do this.

Future Work

All of the following were not implemented due to time constraints, but could significantly improve the functionality of the device.

The group originally intended to allow the visitor to send a text message along with the picture via Morse Code. However, this is generally impractical and would have been more a programming exercise than an added feature.

Implementing a custom interface with the webcam would enable it to take larger pictures and thus give a clearer image of the person ringing the BeagleBell. However, this was beyond the scope of this course and was purposefully avoided as an I/O component.

Although a vague API of sorts is available online for the SMS gateways of many cell carriers, some of them do not work as advertised. In order to resolve this issue we would likely need to contact the carriers' customer support. On further thought, the group decided that the vast majority of homeowners now own smartphones which can easily retrieve e-mails, making this feature somewhat less important.

References

[1] S. Monk. (2013, Jul. 30). *Connecting a Push Button to BeagleBone Black* [Online]. Available: http://learn.adafruit.com/connecting-a-push-button-to-beaglebone-black/overview

[2] M. Darling. (2013, Sep. 24). *How to Achieve 30 fps with BeagleBone Black, OpenCV, and Logitech C920 Webcam* [Online]. Available: http://blog.lemoneerlabs.com/3rdParty/Darling_BBB_30fps_DRAFT.html

Appendix A: Driver Class

```
#!/usr/bin/env python

import time, sys, ConfigParser
import Adafruit_BBIO.GPIO as GPIO
from emailer import Mailer
import webcam as cam

RETRY = 5; WAIT = 30

def main():

	#Import the config file
	config = ConfigParser.RawConfigParser()
	config.read("beaglebell.conf")

	#Get a list of all the addresses in the config file
	sections = config.sections()
	sections.remove("Sender")
	addresses = [config.get(s,"address") for s in sections]

	#Get the Sender info
	server = config.get("Sender","server")
	username = config.get("Sender","username")
	password = config.get("Sender","password")

	#Create a new Mailer
	mailer = Mailer(username,password,addresses,server)

	#Setup the input pin
	GPIO.setup("P8_12", GPIO.IN)

	#Keep track of the last state of the button
	old = 0;

	#Clear Screen
	for i in range(0,20):
	print ""

	while True:

	#Get the new state of the button
	new = GPIO.input("P8_12")

	#If the button was just pressed down
	#take a picture and send an e-mail
	if old==0 and new==1:

		#Take picture
		fname = time.strftime("%c").replace(" ","_").replace(":","-")+".png"
		cam.take_picture(fname)

		#Print lines to hide weird opencv output
		for i in range(0,20):
		print ""

		#E-mail it
		sent = False
		count = 0
		while(sent == False and count < RETRY):
		try:
			mailer.mail(fname)
			sent = True
			print('Ding dong! Message sent!')
		except Exception: #TODO: Be more specific with the exception to catch here...
			sys.stderr.write('Unable to send message. Waiting to retry.\n')
			time.sleep(WAIT)
			count = count + 1
		if count == 5:
		print "Failed to send message. Check your connection."

if __name__ == "__main__":
	main()
```

Appendix B: Button Testing Class

```
#!/usr/bin/env python

# Following this guide on Adafruit:
# http://learn.adafruit.com/connecting-a-push-button-to-beaglebone-black/overview

import Adafruit_BBIO.GPIO as GPIO

def main():

	GPIO.setup("P8_12", GPIO.IN)

	old = 0

	while True:

	new = GPIO.input("P8_12")
	if old==0 and new==1:
		with open("/sys/class/leds/beaglebone:green:usr0/brightness","w") as led0:
		led0.write("1")
	elif old==1 and new==0:
		with open("/sys/class/leds/beaglebone:green:usr0/brightness","w") as led0:
		led0.write("0")

	old = new

if __name__ == "__main__":
	main()
```

Appendix C: Webcam Capture Class

```
#!/usr/bin/env python

import cv2

def take_picture(path):
	cap = cv2.VideoCapture(0) # 0 is device id - first camera on system

	cap.set(3, 320)
	cap.set(4, 240)

	retval, image = cap.read() # Take a picture!
	result = cv2.imwrite(path, image) # Write image to a file

	del(cap) # Close camera
```

Appendix D: Emailer Class

```
#!/usr/bin/env python

# Import smtplib for the actual sending function
import smtplib

# Here are the email package modules we'll need
from email.mime.image import MIMEImage
from email.mime.multipart import MIMEMultipart

', '
class Mailer:
	def __init__(self, username, password, address, host, port = 465):
	self.usr = username   #The doorbell owner's username
	self.to = address   #The destination of the emails.
	self.pas = password        #The password of the doorbell owner.
	self.h = host   #The host and port to use to send the email.
	self.p = port
	def mail(self, file):
	#Sends an email message with a picture attachment in PNG format to the provided address.
	#Creating the email:
	msg = MIMEMultipart()   # Create the container (outer) email message.
	msg['Subject'] = 'Someone rang your doorbell!'
	msg['From'] = self.usr
	msg['To'] = ",".join(self.to)
	msg.preamble = ''

	#Attachting the picture to the email:
	# Assume the provided file is in proper PNG format:
	fp = open(file, 'rb')
	img = MIMEImage(fp.read())   # Open the files using extension to determine type
	fp.close()
	msg.attach(img) #Attach the image to the email.

	# Connect to the SMTP server, login, and send the email.
	server = smtplib.SMTP_SSL(self.h, self.p)
	server.login(self.usr, self.pas)
	server.sendmail(self.usr, self.to, msg.as_string())
	server.quit
```

Appendix E: Sample Config File

```
	[Sender]
	server=smtp.gmail.com
	username=username@gmail.com
	password=password

	[My Cell]
	address=1002003000@vtext.com

	[My e-mail]
	address=zzz@zzz.com
```