Smart Software Project

Lecture: Week 8

Bluetooth Comm. with Android Dev.

Prof. HyungJune Lee

hyungjune.lee@ewha.ac.kr

이화여자대학교
EWHA WOMANS UNIVERSITY

Today

- Review from the last lecture
  - Timer
- Mobile App Programming Tool for Android
  : App Inventor (created by Google)
- ATmega2560 MCU Interrupt
- Announcement

Project Proposal Submission

- Project proposal per team
  - Due: 5pm on April 22, Friday
  - What to submit:
    - 1) Report hardcopy (printed report)
    - 2) Report softcopy (report file submission to Cyber Campus)
  - Where to submit:
    - HW box at "HyungJune Lee" in front of Asan 221-1
  - Language: English or Korean
  - Start discussing what your team will do with your team partner

Project Proposal

- Project name (in English)
- Project statement
  - Goals of your project
- Project description
  - What will your project be doing?
  - Key functions
- Contributions of your work to research or industry
- Related work
  - Any existing previous works (at least two) similar to your project
  - What's similar and different?
- System overview & architecture
  - Block diagram of main blocks
  - What each block is doing
  - How each block is connected to other blocks, i.e., interface
    - Any message is exchanging between blocks? e.g., request/reply, data, etc?

Project Proposal

- Development environment
  - Arduino Mega 2560-based SmartCAR (or others if any, e.g., Android device)
  - Androx Studio IDE (or others if any)
  - What kind of sensors are to be used in your project
  - Other information from the connection to Android device or other devices? (location, Internet, etc.)
- Verification procedure
  - How can you test if your project works properly as designed
  - Test cases
- What do you anticipate will be the easiest part of your project?
- What do you anticipate will be the most difficult part of your project?
- Detailed time plan
- References

Evaluation criteria

- Format requirement
  - 5 points
- Creativity
  - 5 points
- Clarity
  - 5 points
- Concreteness (of software architecture)
  - 5 points
- Implementability
  - 5 points

- Total score: 25 points

Class Schedule

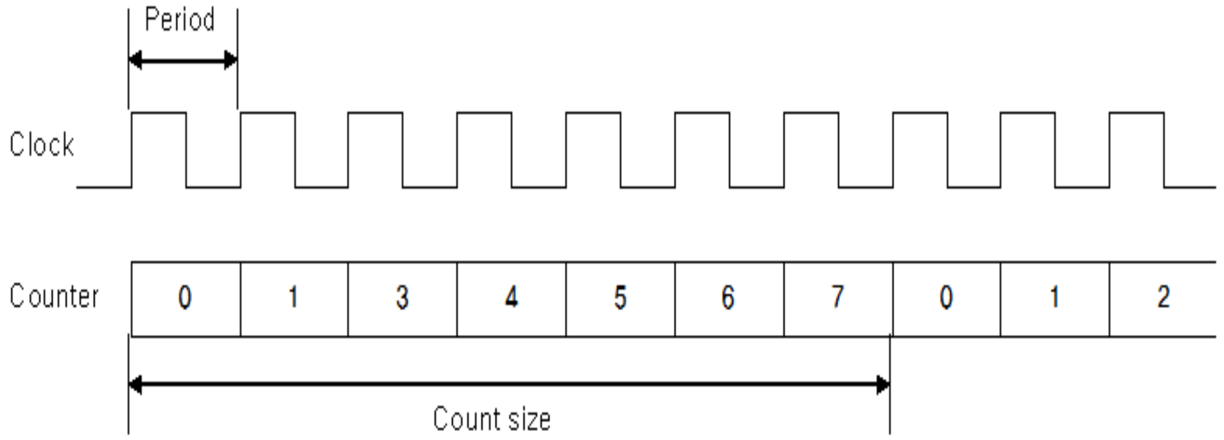
Week	Lecture Contents	Lab Contents
Week 1	Course introduction	Arduino introduction: platform & programming environment
Week 2	Embedded system overview & source management in collaborative repository (using GitHub)	Lab 1: Arduino Mega 2560 board & SmartCAR platform
Week 3	ATmega2560 Micro-controller (MCU): architecture & I/O ports, Analog vs. Digital, Pulse Width Modulation	Lab 2: SmartCAR LED control
Week 4	Analog vs. Digital & Pulse Width Modulation	Lab 3: SmartCAR motor control (Due: HW on creating project repository using GitHub)
Week 5	ATmega2560 MCU: memory, I/O ports, UART	Lab 4: SmartCAR control via Android Bluetooth
Week 6	ATmega2560 UART control & Bluetooth communication between Arduino platform and Android device	Lab 5: SmartCAR control through your own customized Android app (Due: Project proposal)
Week 7	Midterm exam	
Week 8	ATmega2560 Timer, Interrupts & Ultrasonic sensors	Lab 6: SmartCAR ultrasonic sensing
Week 9	Infrared sensors & Buzzer	Lab 7: SmartCAR infrared sensing
Week 10	Acquiring location information from Android device & line tracing	Lab 8: Implementation of line tracer
Week 11	Gyroscope, accelerometer, and compass sensors	Lab 9: Using gyroscope, accelerometer, and compass sensors
Week 12	Project	Team meeting (for progress check)
Week 13	Project	Team meeting (for progress check)
Week 14	Course wrap-up & next steps	
Week 15	Project presentation & demo I (Due: source code, presentation slides, & poster slide)	Project presentation & demo II
Week 16	Final week (no final exam)	

Today

- Review from the last lecture
  - **Timer**
- Mobile App Programming Tool for Android
  : App Inventor (created by Google)
- ATmega2560 MCU Interrupt
- Announcement

Timer: Clock & Counter

- Clock
  - Fundamental "Clock" in computer system
  - Generated by an oscillator crystal
  - Period: one cycle of '1' and '0'
  - Counter: counts the number of clock cycles



이화여자대학교
EWHA WOMANS UNIVERSITY

ATmega2560 Timers

- 0 ~ 5: Total 6 timers
- Timer0, Timer2
  - 8 bit counter (count 0~255)
- Timer1, Timer3, Timer4, Timer5
  - 16 bit counter (count 0~65535)

ATmega2560 Timers

- Timerx function (x =0~5)
  - Timerx library function: use timer and counter x
  - Timer0 & Timer2
    - 8-bit timer: count 0 ~ 255, and then when it comes to 0, it will generate a timer overflow interrupt
  - All other timers, Timer1, Timer3, Timer4, & Timer5
    - 16-bit timer: count 0 ~ 65535, and then when it comes to 0, it will generate a timer overflow interrupt

ATmega2560 Timers

- 3 Functions
  - Timerx::set(unsigned long us, void (*f)())
    - Set an timer period & Interrupt Service Routine function
      - us: interrupt period in micro second
      - void(*f)(): Interrupt Service Route to execute when an timer overflow interrupt occurs
  - Timerx::start()
    - Start the timer
  - Timerx::stop()
    - Stop the timer

ATmega2560 Timers

```
#include <Timer2.h>

#define FRONT_LED   10

int LED_state = 0;

void Timer2_ISR() {
    digitalWrite(FRONT_LED, LED_state);
    if (LED_state)
        LED_state = 0;
    else
        LED_state = 1;
}

void setup() {
    pinMode(FRONT_LED, OUTPUT);
    Timer2::set(1000000, Timer2_ISR);
    Timer2::start();
}

void loop() {
}
```

- What is this program doing?

Delay vs. Timer Approach

```
#define FRONT_LED    10

int LED_state = 0;

void setup() {
    pinMode(FRONT_LED, OUTPUT);
}

void loop() {
    digitalWrite(FRONT_LED, LED_state);
    delay(1000);

    if (LED_state)
        LED_state = 0;
    else
        LED_state = 1;
}
```

```
#include <Timer2.h>

#define FRONT_LED    10

int LED_state = 0;

void Timer2_ISR() {
    digitalWrite(FRONT_LED, LED_state);
    if (LED_state)
        LED_state = 0;
    else
        LED_state = 1;
}

void setup() {
    pinMode(FRONT_LED, OUTPUT);
    Timer2::set(1000000, Timer2_ISR);
    Timer2::start();
}

void loop() {
}
```

- Which one is better in terms of what?

Today

- Review from the last lecture
  - Timer

- **Mobile App Programming Tool for Android**
  : App Inventor (created by Google)

- ATmega2560 MCU Interrupt

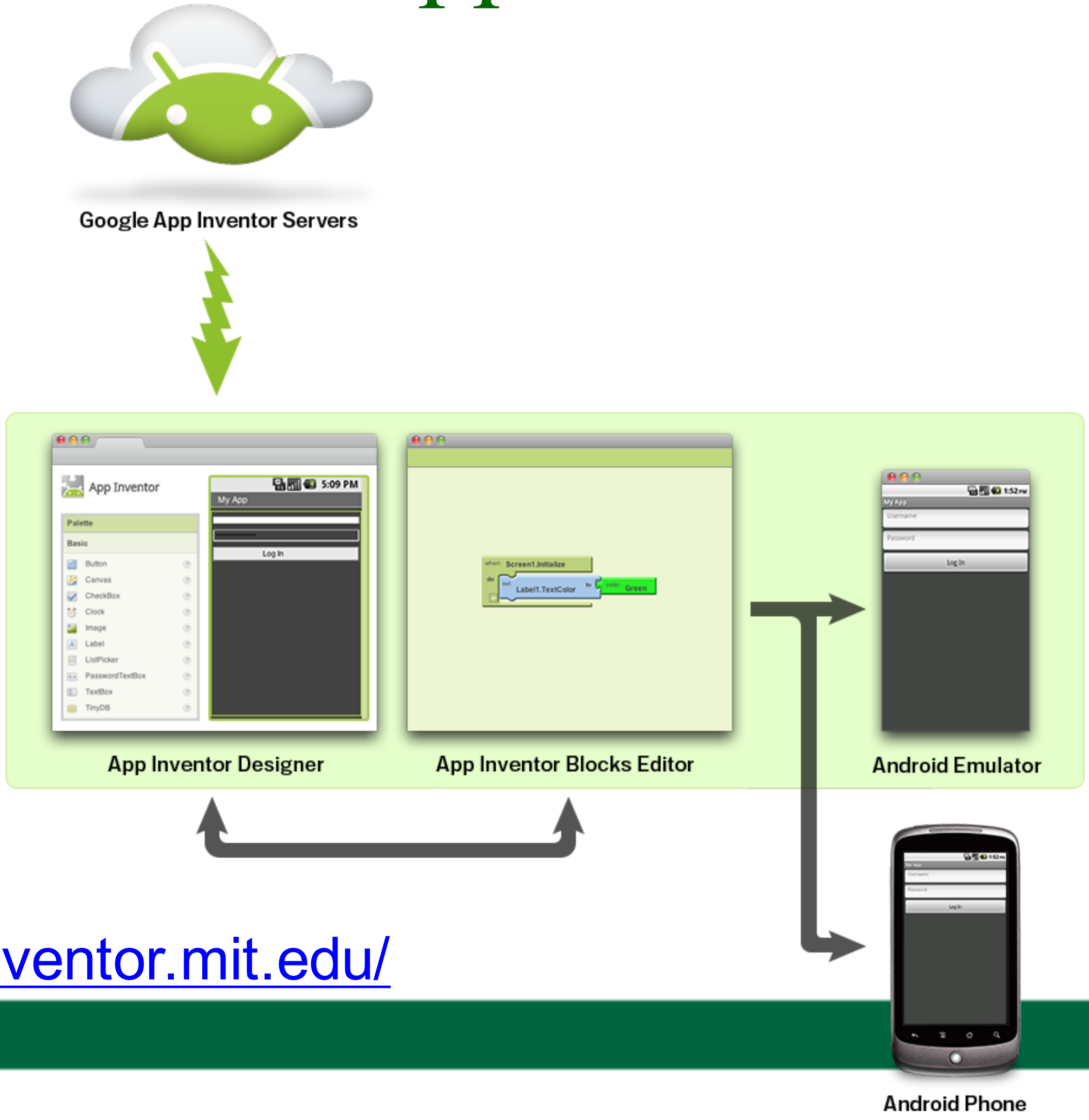
- Announcement

App Development depending on OS

- iPhone
  - Swift (Previously Objective C)
  - High level tools to turn pre-existing content into an app. (e.g., http://www.appmakr.com/learn_more/)

- Android
  - Java
  - App Inventor (created by Google)

What is App Inventor?



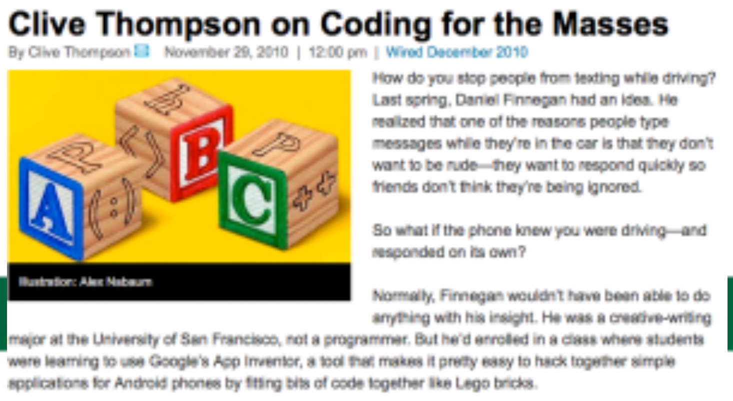
http://ai2.appinventor.mit.edu/

App Inventor

- Blocks language, like plugging in puzzle pieces
- Allows fast prototyping for sketching out app ideas
- Democratizes app building
- Video clip
  - http://www.youtube.com/watch?v=E9Zm56Od1pw

Ex) "No Texting While Driving"

Daniel Finnegan.English Major



Clive Thompson on Coding for the Masses

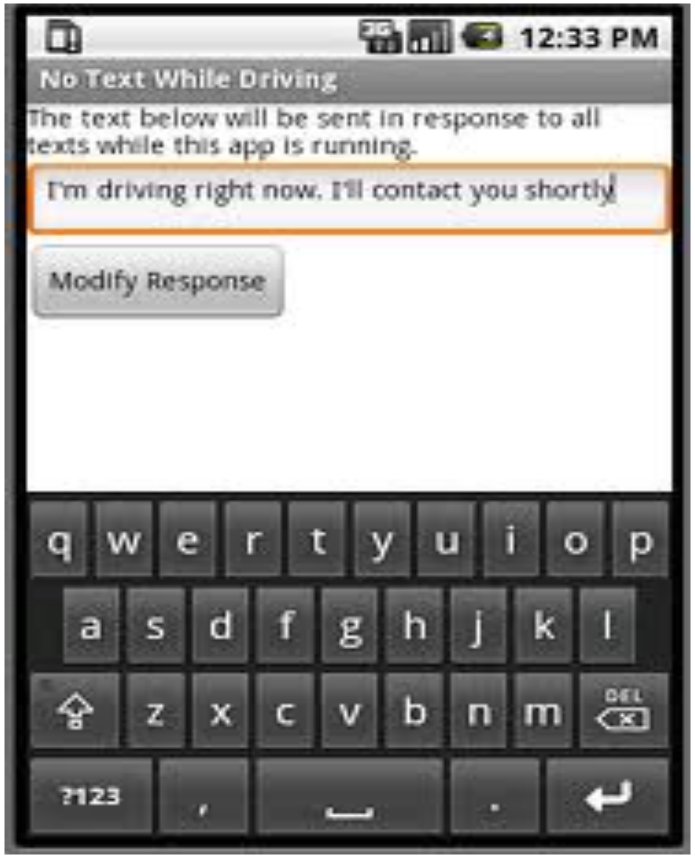
By Clive Thompson | November 29, 2010 | 12:00 pm | Wired December 2010

Illustration: Alex Nabaum

How do you stop people from texting while driving? Last spring, Daniel Finnegan had an idea. He realized that one of the reasons people type messages while they're in the car is that they don't want to be rude—they want to respond quickly so friends don't think they're being ignored.

So what if the phone knew you were driving—and responded on its own?

Normally, Finnegan wouldn't have been able to do anything with his insight. He was a creative-writing major at the University of San Francisco, not a programmer. But he'd enrolled in a class where students were learning to use Google's App Inventor, a tool that makes it pretty easy to hack together simple applications for Android phones by fitting bits of code together like Lego bricks.



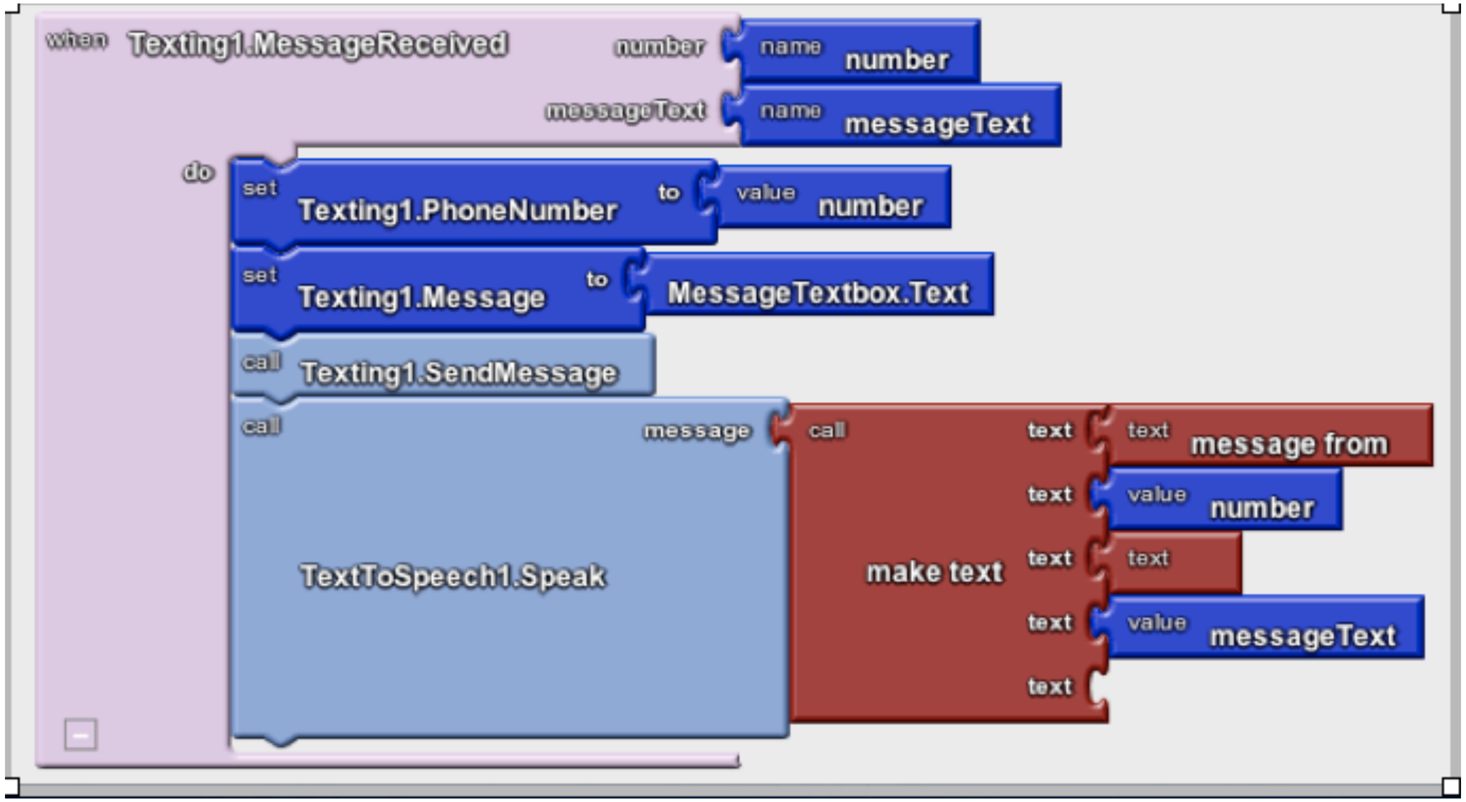
12:33 PM

No Text While Driving

The text below will be sent in response to all texts while this app is running.

I'm driving right now. I'll contact you shortly

App Blocks using App Inventor



```
when Texting1.MessageReceived   number       name number
                                messageText  name messageText
  do  set Texting1.PhoneNumber to  value number
      set Texting1.Message to  MessageTextbox.Text
      call Texting1.SendMessage
      call TextToSpeech1.Speak
           message  call make text  text  text message from
                                    text  value number
                                    text  text
                                    text  value messageText
                                    text
```

What do you think this app does?

App Inventor History

- Summer 2009
  - Hal Abelson at MIT Media Lab and Google
  - Pilot program with 10 schools
  - Public launch in July 2010
- App Inventor 2 is launched on Dec 3, 2013
- App Inventor 1 (as of Dec 2013)
  - 1.3 million users
  - 3.2 million apps
- App Inventor 2 (as of Dec 2013)
  - 100,000 users
  - 140,000 apps

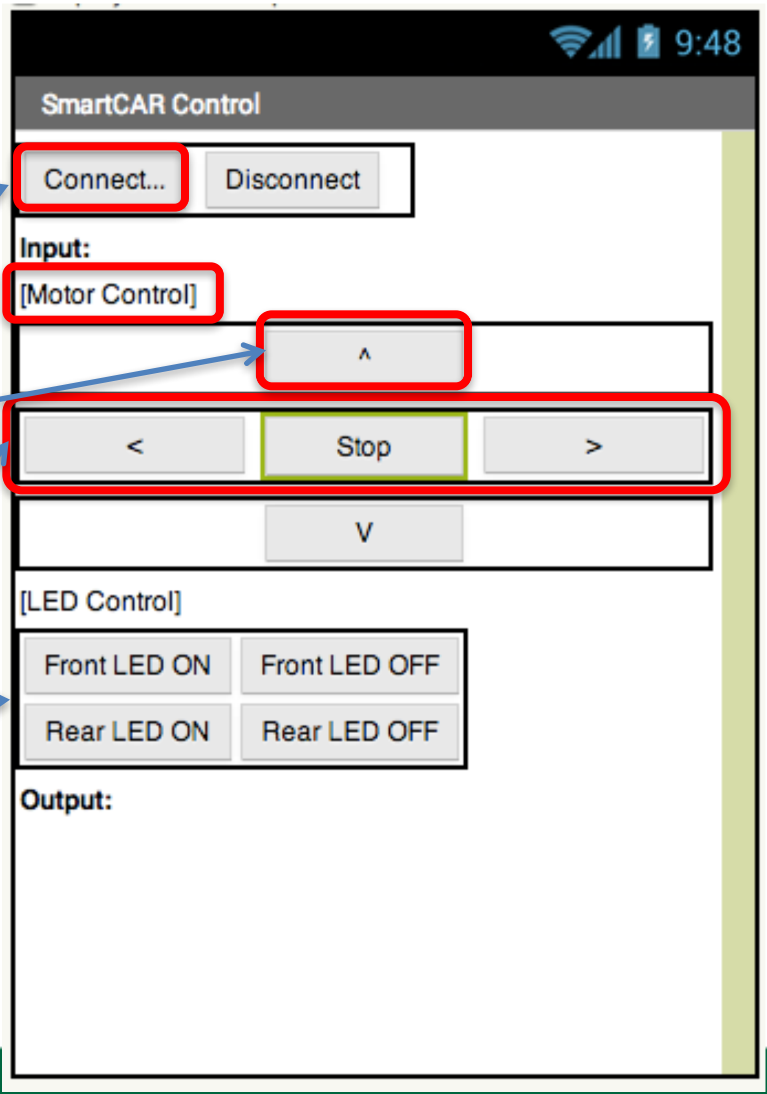
Why is it so easy?

- **No syntax**
  - The blocks language eliminates the need to remember and type code
- **Everything is right in front of you**
  - Components and functions are in drawers
  - Just find, drag, and drop
- **Events at top level**
  - "When this happens, the app does this" is the correct conceptual model
- **High-level components**
  - The app inventor team has built a great library with simplicity the main goal
- **Only some blocks plug-in**
  - You can't do things that don't make sense
- **Concreteness**
  - Less abstract than many languages

SmartCAR Control App

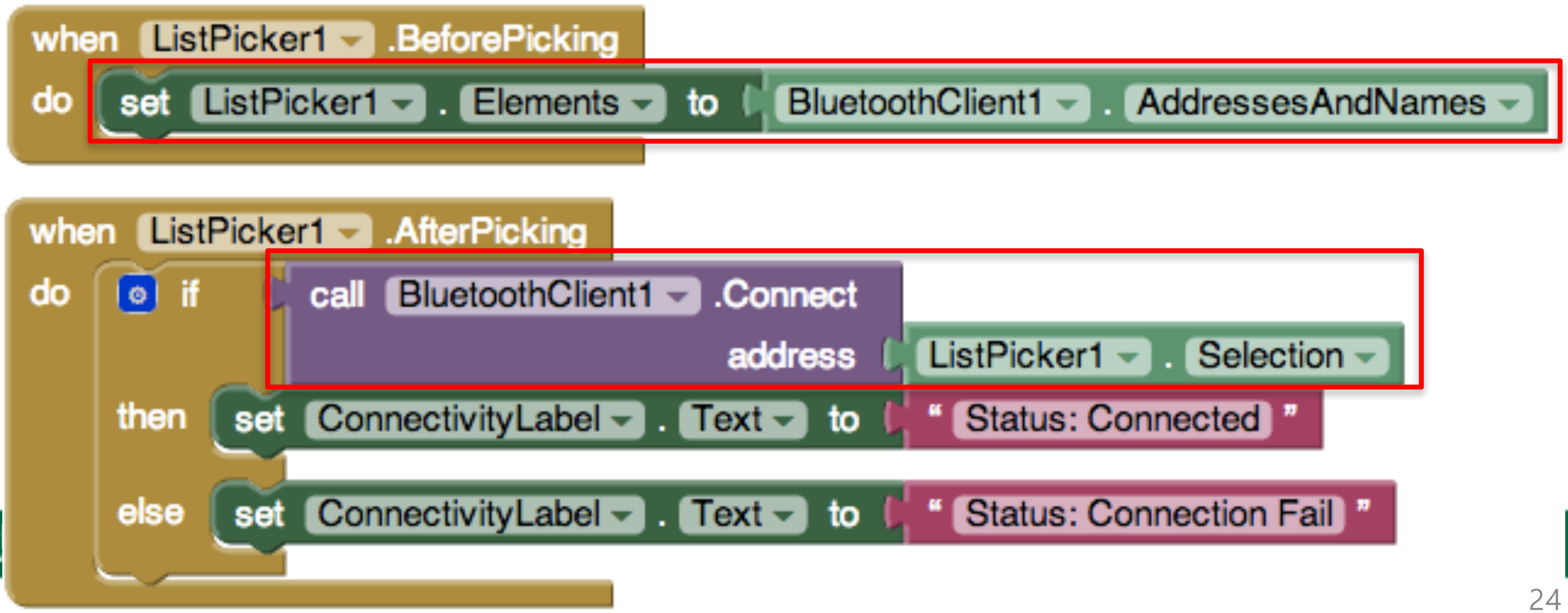
- http://ai2.appinventor.mit.edu
- Click on "New Project"
- Enter "SmartCAR" in Project Name (One word, no space)

- Under "User Interface"
  - Drag-and-drop "ListPicker" component
    - To select a Bluetooth device
  - Drag-and-drop "Button" component
  - Drag-and-drop "Label" component
- Under "Layout"
  - Drag-and-drop "HorizontalArrangement"
  - Drag-and-drop "TableArrangement"
- Under "Connectivity"
  - Drag-and-drop "BluetoothClient"
    - **Uncheck "Secure"**
- Under "Sensors"
  - Drag-and-drop "AccelerometerSensor"



Bluetooth Connection

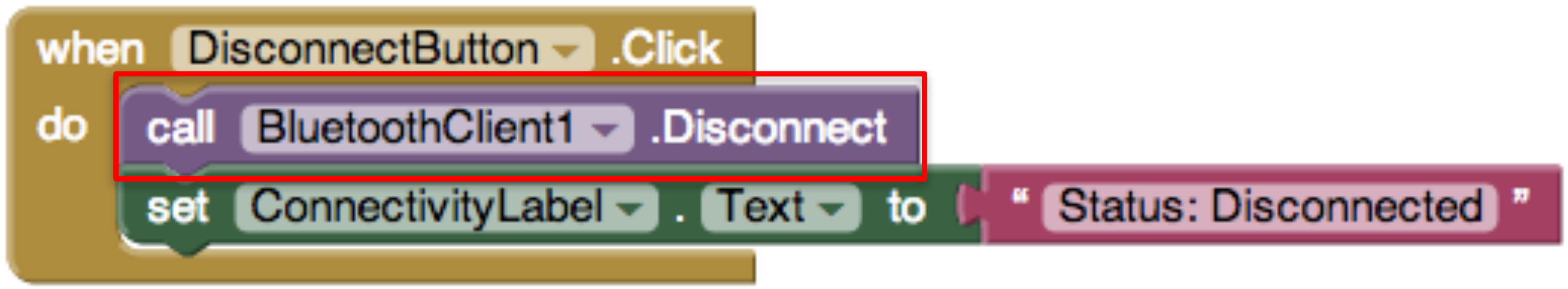
- Before picking a Bluetooth device in ListPicker
  - Show all connectable Bluetooth devices' Addresses and Names in Element
- After picking a Bluetooth device in ListPicker
  - Connect to the device selected in ListPicker.Selection
    - If success, print "Status: Connected" in label
    - If fail, print "Status: Connection Fail" in label



```
when ListPicker1 .BeforePicking
do  set ListPicker1 . Elements to  BluetoothClient1 . AddressesAndNames

when ListPicker1 .AfterPicking
do  if    call BluetoothClient1 .Connect
                        address  ListPicker1 . Selection
    then  set ConnectivityLabel . Text to  " Status: Connected "
    else  set ConnectivityLabel . Text to  " Status: Connection Fail "
```

Bluetooth Disconnection

- When “Disconnect” button is clicked
  - Execute BluetoothClient1.Disconnect
  - Print “Status: Disconnected” in label

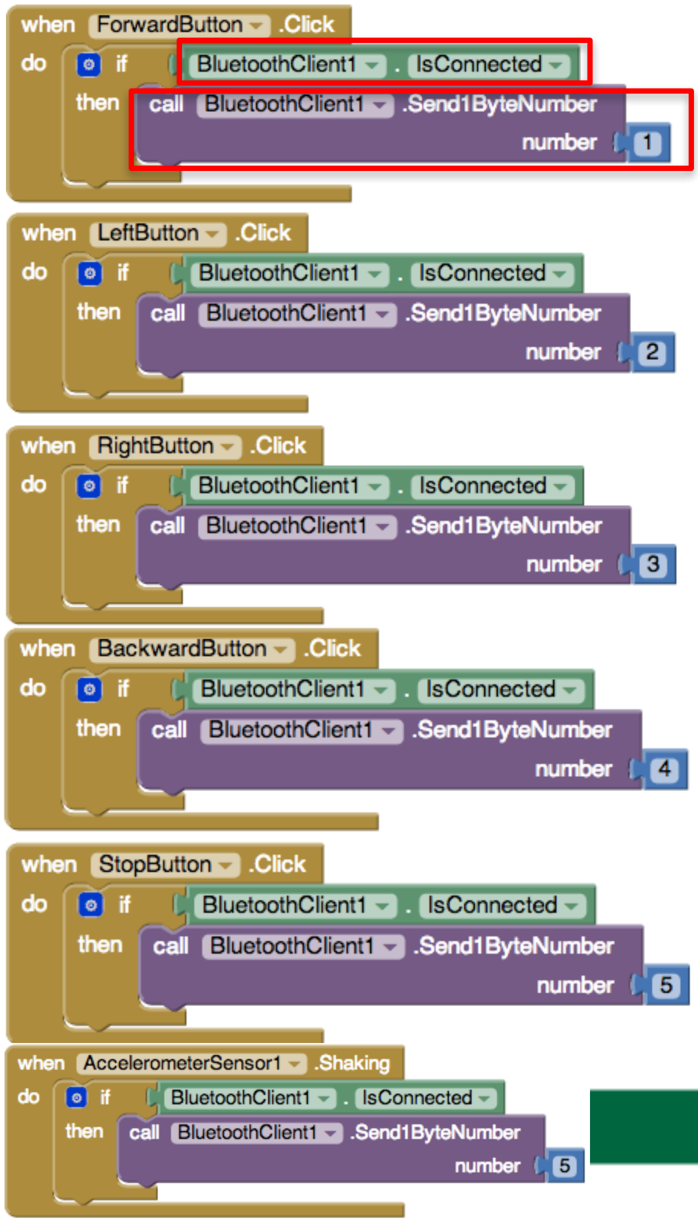


```
when DisconnectButton .Click
do  call BluetoothClient1 .Disconnect
    set ConnectivityLabel . Text to " Status: Disconnected "
```

SmartCAR Motor Control

- Move forward (command byte: 1)
  - Send "1" in number using "BluetoothClient.Send1ByteNumber"
- Turn left (command byte: 2)
  - Send "2" in number using "BluetoothClient.Send1ByteNumber"
- Turn right (command byte: 3)
  - Send "3" in number using "BluetoothClient.Send1ByteNumber"
- Move backward (command byte: 4)
  - Send "4" in number using "BluetoothClient.Send1ByteNumber"

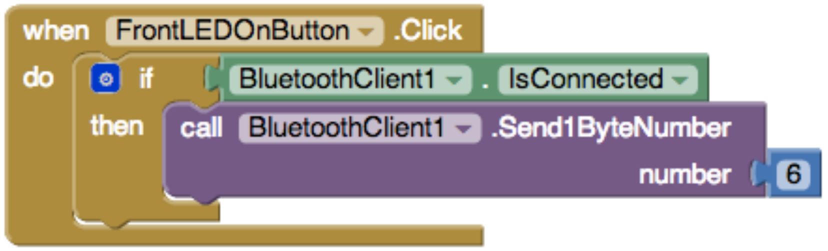
- Stop (command byte: 5)
  - Send "5" in number using "BluetoothClient.Send1ByteNumber"



```
when ForwardButton .Click
do  if   BluetoothClient1 . IsConnected
    then call BluetoothClient1 .Send1ByteNumber
                                number  1

when LeftButton .Click
do  if   BluetoothClient1 . IsConnected
    then call BluetoothClient1 .Send1ByteNumber
                                number  2

when RightButton .Click
do  if   BluetoothClient1 . IsConnected
    then call BluetoothClient1 .Send1ByteNumber
                                number  3

when BackwardButton .Click
do  if   BluetoothClient1 . IsConnected
    then call BluetoothClient1 .Send1ByteNumber
                                number  4

when StopButton .Click
do  if   BluetoothClient1 . IsConnected
    then call BluetoothClient1 .Send1ByteNumber
                                number  5

when AccelerometerSensor1 .Shaking
do  if   BluetoothClient1 . IsConnected
    then call BluetoothClient1 .Send1ByteNumber
                                number  5
```

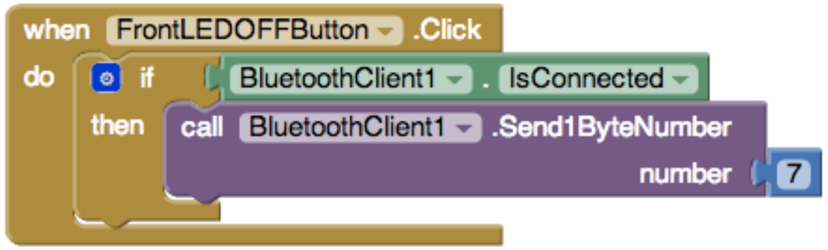
SmartCAR LED Control

- Turn Front LED ON (command byte: 6)
  - Send "6" in number using "BluetoothClient.Send1ByteNumber"



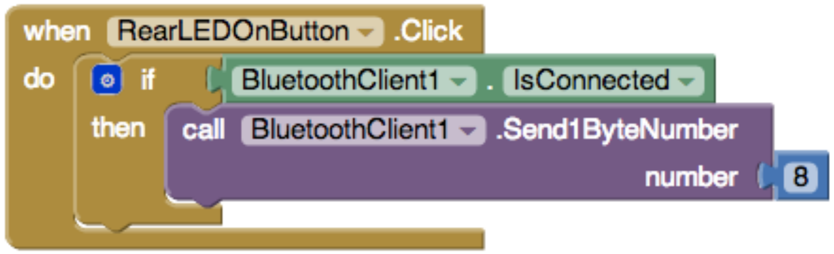
```
when FrontLEDOnButton .Click
do  if BluetoothClient1 . IsConnected
    then call BluetoothClient1 .Send1ByteNumber
                                 number 6
```

- Turn Front LED OFF (command byte: 7)
  - Send "7" in number using "BluetoothClient.Send1ByteNumber"



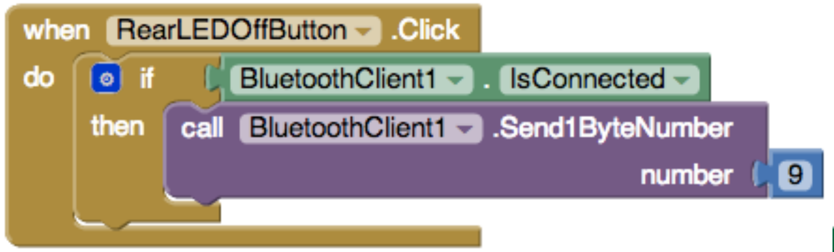
```
when FrontLEDOFFButton .Click
do  if BluetoothClient1 . IsConnected
    then call BluetoothClient1 .Send1ByteNumber
                                 number 7
```

- Turn Rear LED ON (command byte: 8)
  - Send "8" in number using "BluetoothClient.Send1ByteNumber"



```
when RearLEDOnButton .Click
do  if BluetoothClient1 . IsConnected
    then call BluetoothClient1 .Send1ByteNumber
                                 number 8
```

- Turn Rear LED OFF (command byte: 9)
  - Send "9" in number using "BluetoothClient.Send1ByteNumber"



```
when RearLEDOffButton .Click
do  if BluetoothClient1 . IsConnected
    then call BluetoothClient1 .Send1ByteNumber
                                 number 9
```

SmartCAR Firmware

```
unsigned char text[] = "₩r₩n Welcome! Arduino Mega₩r₩n UART Test Program.₩r₩n";

void setup()
{
// Add your initialization  code here
    int i = 0;
    Serial.begin(115200);

    while (text[i] != '₩0')
        Serial.write(text[i++]);

    Serial.write("Received  cmds: ");

    //initialize ports
    pinMode(....);
    ....
    digitalWrite(...);
}
```

SmartCAR Firmware

```
void loop()
{
    if (Serial.available() > 0)
    {
        int command = Serial.read();
       Serial.print(command, DEC);
        Serial.print(" ");
        switch (command)
        {
            case 1:
                move_stop();
                delay(500);

                move_forward();
                break;
            case 2:
                move_stop();
                delay(500);

                turn_left();
                break;
            case 3:
                move_stop();
                delay(500);

                turn_right();
                break;
            case 4:
                move_stop();
                delay(500);

                move_backward();
                break;
```

```
            case 5:
                move_stop();
                break;
            case 6:
                front_led_control(true);
                break;
            case 7:
                front_led_control(false);
                break;
            case 8:
                rear_led_control(true);
                break;
            case 9:
                rear_led_control(false);
                break;
            default:
                move_stop();
                front_led_control(false);
                rear_led_control(false);
        }
    }
}
```

Today

- Review from the last lecture
  - Timer
- Mobile App Programming Tool for Android
  : App Inventor (created by Google)
- **ATmega2560 MCU Interrupt**
- Announcement

Interrupts

- Definition
  - An event external to the currently executing process that causes a change in the normal flow of instruction execution
  - Usually generated by hardware devices external to the CPU
  - Key point: Interrupts are asynchronous w.r.t. current process
    - Typically indicate that some device needs service

Why Interrupts?

- People like connecting devices
  - A computer is much more than the CPU
    - Keyboard, mouse, screen, disk drives
    - Scanner, printer, sound card, camera, etc.
  - These devices occasionally need CPU service
    - But we can't predict when
  - External events typically occurs on a macroscopic timescale
    - We want to keep the CPU busy between these events
- Need a way for CPU to find out devices need attention

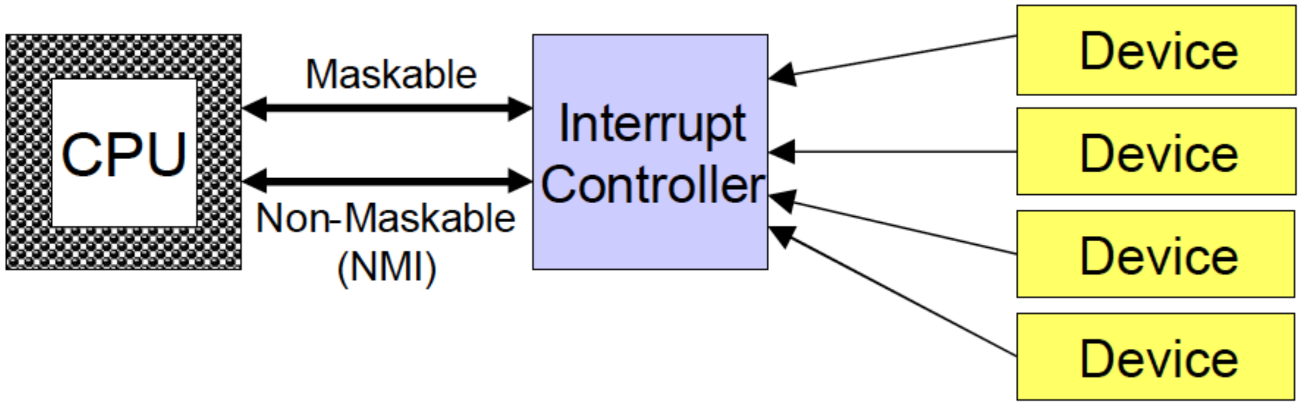
Possible Solution: Polling

- CPU periodically checks each device to see if it needs service
  - Takes CPU time even when no requests pending ☹
  - Overhead may be reduced at expense of response time ☹
  - Can be efficient if events arrive rapidly ☺

"Polling is like picking up your phone every few seconds to see if you have a call…"

Alternative: Interrupts

- Give each device a wire (interrupt line) that it c an use to signal the processor
  - When interrupt is signaled, processor executes a routine called an interrupt handler to deal with the interrupt
  - No overhead when no requests pending



Polling vs. Interrupts

- Polling
  - "Like picking up your phone very few seconds to see if you have a call"
- Interrupts
  - "Like waiting for the phone to ring"

- Interrupts win if processor has other work to do and event response time is not critical
- Polling can be better if processor has to respond to an event ASAP
  - May be used in device controller that contains dedicated secondary processor

Hardware Interrupt Handling

- Details are architecture dependent
- Interrupt controller signals CPU that interrupt has occurred, passes interrupt number
  - Interrupts are assigned priorities to handle simultaneous interrupts
  - Lower priority interrupts may be disabled during service
- CPU senses (checks) interrupt request line after every instruction; if raised, then:
  - Uses interrupt number to determine which handler to start
  - Interrupt vector associates handlers with interrupts
- Basic program state saved (as for system call)
- CPU jumps to interrupt handler
- When interrupt is done, program state reloaded and program resumes

Arduino Interrupt Handling

- ATmega2560 Interrupt trigger
  - When input signal has changed from '0' to '1' or from '1' to '0' (edge trigger), or
    stays at 1 or 0 (level trigger), an interrupted is "triggered"

    - Edge Trigger
      - At the moment that changes from '1' to '0' (Falling Edge Trigger)
      - At the moment that changes from '0' to '1' (Rising Edge Trigger)
      - Pulse should stay at least 50ns

    - Level Trigger
      - If input signal stays for a moment, then it is triggered

Arduino Interrupt Functions

- Two interrupt-related functions supported in Arduino
  - attachInterrupt(interrupt, function, mode)
    - Set an interrupt number and triggering way
      - Interrupt: Interrupt number to use.
      - Function: Interrupt Service Routine function upon interrupt occurred
      - Mode: Interrupt mode
        - » "Level Trigger"
          - LOW – Triggered at the LOW level
          - CHANGE – Triggered if the level has been changed
        - » "Edge Trigger"
          - RISING – Triggered at the rising edge
          - FALLING – Triggered at the falling edge
  - detachInterrupt(interrupt)
    - Terminate the usage of interrupt
      - Interrupt: Interrupt number to stop using the interrupt

  - Interrupt information in SmartCAR

Interrupt No.	0	1	2	3	4	5	6	7
Port / Pin No.	PE4 / 2	PE5 / 3	PD0 / 21	PD1 / 20	PD2 / 19	PD3 / 18	PE6 / -	PE7 / -

Sample Program

```
#define MY_PIN  13
int state = LOW;

void setup(){
   pinMode(MY_PIN, OUTPUT);
    attachInterrupt(0, blink, CHANGE);
}

void loop() {
   digitalWrite(MY_PIN, state);
}

void blink() {
   state = !state;
}
```

- attachInterrupt(0, blink, CHANGE)
  - Interrupt setting
    - 1st argument
      - Use interrupt number 0 (pin 2) for interrupt (assuming that MY_PIN 13 is also connected to pin 2)
    - 2nd argument
      - Interrupt Service Routine function you want to execute upon interrupt occurred
      - Called as "interrupt handler": when an interrupt occurs, blink function will be called
    - 3rd argument
      - CHANGE: whenever pin level has changed (high to low or low to high), the interrupt occurs

Course Announcement

- For lab session, we will cover
  - SmartCAR Bluetooth Communication with Android device

- Next week, we will learn
  - Ultrasonic sensors
  - Infrared sensors