Arduviz, A Visual Programming IDE for Arduino

Adin Baskoro Pratomo, Riza Satria Perdana
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
13513058@std.stei.itb.ac.id, riza@stei.itb.ac.id

***Abstract*—Arduino is an open source computing platform in a form of single-board microcontroller. The microcontroller in Arduino is reprogrammable. Officially supported way to program Arduino is by using Arduino language and Arduino IDE. Another way to program an Arduino board is by using visual programming approach. Language used in visual programming approach is called Visual Programming Language. Commonly used existing tools that enable a programmer to write Arduino program visually are Ardublock and miniBloq. Both of those tools have their own strength. But, because those are separate tools, a programmer can't use all of those strengths to create a program. We have implemented Arduviz, a visual programming integrated development environment. Arduviz has most of advantages from both Arduviz and miniBloq such as instant code generation and stand alone development environment.**

Keywords—visual programming; Arduino; integrated development environment;

I. INTRODUCTION

Arduino is an open source computing platform for creating interactive objects in a form of single-board microcontroller[1]. Most Arduino boards use an Atmel 8-bit AVR microcontroller as the processor. The microcontroller already includes flash memory to store program, RAM, and EEPROM for non-volatile data storage. An Arduino board exposes various I/O pins on the microcontroller for use by other circuits. Microcontroller used in an Arduino is reprogrammable. To upload a program to an AVR microcontroller, external programmer is needed. External programmer is a hardware that enable computer to communicate with the microcontroller in a system. However, the microcontroller in an Arduino board is preprogrammed with a software called bootloader that can write program to the flash memory without external programmer. There are many different types of Arduino board with different specification, such as external connectivity and microcontroller used.

Officially supported way to write a program for Arduino is by Arduino programming language and Arduino IDE[2]. Arduino programming language is based on Processing, an imperative open source programming language. Its syntax is similar to C++, with some additional predefined functions and constants. A program written in Arduino IDE is called sketch. Minimal Arduino sketch consist of two functions; *setup()* and *loop()*. Function *setup()* is called once when program starts after power-up or board reset. It's usually used to initialize variables, libraries, or I/O pins. After the *setup()* has been called, *loop()* is called and executed repeatedly until the board is powered or is reset[3].

To run the sketch on an Arduino board, it must first be compiled. In Arduino IDE, before passing the code to the compiler the sketch is transformed to a valid C++ source code. The transformation includes definitions needed for standard Arduino program. Then, the IDE searches for function definitions and create declaration for them. Then, the sketch is compiled by *avr-gcc*, a compiler that can target AVR microprocessors. Result of compilation process is a binary file in Intel hex format. The binary file then can be uploaded to the Arduino board to be run[4].

Another way to program an Arduino board is by using visual programming method. Visual programming is a programming method that allows programmer to program by manipulating graphics, instead of writing text codes. Language used in visual programming approach is called Visual Programming Language (VPL). VPL is a language that has its structure represented in pictorial notation, which include icon, color, texture, and other non-textual device. In traditional programming language, structure and data are coded in strings, requiring extra layer of syntax. By expressing structure and data pictorially, a more concrete representation might be achieved, and thus make it easier to build and debug program[5]. This can be seen from visual programming languages that is used to teach programming such as ALICE 3 (https://www.alice.org/) and Scratch (https://scratch.mit.edu/). Two examples of VPL for programming Arduino system are miniBloq (http://blog.minibloq.org/) and Ardublock (https://github.com/taweili/ardublock).

Ardublock is implemented as add-on to Arduino IDE. A programmer assemble blocks to make a program that can be run in an Arduino system using Ardublock. Program structure in Ardublock is very similar to Arduino language. Beside standard I/O, Ardublock includes support for advanced features available in an Arduino board such as SPI communication. Program made in Ardublock is converted to Arduino language and displayed in Arduino IDE just before compilation. Compilation process is exactly the same as when using Arduino IDE alone. Ardublock is still actively developed and can be accessed via its GitHub page.

MiniBloq in other hand is a stand alone IDE with its own environment. Like Ardublock it also use blocks as representation of structure and data. The structure is quite similar with Arduino language with few differences. Arduino features supported in Minibloq is less than Ardublock. Unlike Ardublock, miniBloq generates C++ code as opposed to Arduino language code in Ardublock. It uses its own Arduino library and environment, so it doesn't depend on Arduino IDE. Compilation and upload

978-1-5386-1449-5/17/$31.00 ©2017 IEEE

process is also done inside the miniBloq IDE. miniBloq hasn't been updated for few years and the development is stalled.

Both Ardublock and miniBloq still have room for improvement in support for Arduino features, code generation, and development environment. More support for Arduino features is always good, as programmer can use the VPL and IDE to create more complex system. While one can argue code generation is not very important because the generated code will only be passed to the compiler to produce executable file, It can be used to give more insight about how the VPL work. A good development environment can help programmer to focus on building program, instead of spending time configuring the environment.

The rest of this paper is organized as follows: Chapter II explains what is the problem on the existing visual programming tools for Arduino. Chapter III explores existing visual programming language and tools for Arduino. Chapter IV presents design, implementation and testing of proposed solution to create visual programming environment for Arduino that suitable to help people learn Arduino language. Finally, chapter V concludes our research and discusses future improvements of this solution.

II. Problem Statement and Proposed Solution

Ardublock and miniBloq has each own strength and weakness in form of language structure, support of Arduino features, code generation, and development environment. One can't use strengths from both tools to create a program. For example, Ardublock already support many features of Arduino, including advanced features such as serial communication. However, the code generation in Ardublock is still lacking, as it doesn't generate code in real time. Real time code generation is useful for people to learn the relation between block they've assembled with the equivalent code in Arduino language. miniBloq on the other hand, has real time code generation, but with less complete Arduino feature support.

To combine strengths of Ardublock and miniBloq, we have to design an integrated development environment that includes language that can utilize most features in an Arduino board. Compiler and uploader also included in the environment and can be used without manual configuration. Beside basic features like save and load project, serial monitor is also a helpful addition, as it can help a programmer to develop and debug using serial communication between arduino and PC. The IDE also generate code in real time, as the programmer assemble the blocks. The code also can be exported, so it can be opened in Arduino IDE for further editing if needed.

III. Related Works

A. Visual Programming Language Design and Structure

Ardublock represent structure and data as block. Blocks in Ardublock is modeled closely to built-in functions in Arduino language with some abstraction. For example, to set a pin in an Arduino board to output a value, it can be achieved in just one block, whereas in Arduino language, it needs two function calls. Blocks in Ardublock have connection with different shape for different kind of things to prevent connecting the block wrongly. There also different block color to differentiate blocks. The blocks also have label that indicates the function of the block. Ardublock have similar imperative structure like Arduino language. The blocks are assembled from top to bottom, with the top block executed first. It also use same basic structure as Arduino language by using setup and loop block as base for other blocks to connect.

miniBloq also represents structure in blocks. The blocks are assembled in imperative way from top to bottom with top block executed first. . Blocks in miniBloq doesn't have text label. Instead, it uses icon to represent the information about the function. There's tooltip that provide short information about the block when hovered. miniBloq doesn't use setup and loop block like Ardublock. Instead, the programmer must define the loop himself. If no loop given, the program will only execute once

B. Support for Arduino Features

Ardublock support many Arduino features, from simple ones like digital and analog I/O to more advanced feature such as serial communication and writing data to EEPROM. Some of generic hardware such as servo and ultrasonic sensor are already supported by Ardublock. Beside Arduino itself, Ardublock also support various derived *platform* and hardware that use or based on Arduino. In other hand, miniBloq also support basic features of Arduino just as Ardublock does. It also support serial communication. Some Arduino features found in Ardublock is missing in miniBloq, such as writing data to EEPROM. miniBloq supports various derived hardware from Arduino, just like Ardublock.

C. Code Generation, Compilation, and Upload process.

In Ardublock, assembled blocks of code is translated to code in Arduino language. The code is generated when the programmer choose to compile and upload the program. The compilation and upload process is exactly the same as Arduino IDE, because it use Arduino IDE for that purpose..

miniBloq generate C++ code from assembled blocks. It use its own Arduino library to access functionality of Arduino board. Unlike Ardublock, it generates the code in real time, as the block is placed. By using real time code generation, the user can see relation between blocks and the generated code. The code is compiled using *avr-gcc* and uploaded using miniBloq's own tool. Compilation and upload process are started when the programmer wants to.

D. Integrated Development Environment

Ardublock is an add-on to Arduino IDE. Therefore, it shares all tools available in Arduino IDE. The setup process is not complicated, but not easy either. There's no automated process to set up Ardublock. Also because Ardublock depends on Arduino IDE, the version of the IDE used must match Ardublock's requirement. This may be an issue when Arduino IDE is updated. In other hand, miniBloq has its own IDE, with bundled compiler, uploader and device driver. It also have helpful tools like serial monitor. miniBloq comes in form of installer, so the setup process is also very simple.

IV. ARDUVIZ

In this research, we made a new visual programming development environment for programming Arduino called Arduviz. Creating a program using Arduviz is divided into three main process: assembling blocks, transforming the blocks into code in Arduino language, and compilation and upload. Blocks are assembled by dragging a block from toolbox and connect it to another block. Transformation is done by mapping each block with equivalent counterpart in Arduino language. Compiler then takes generated code as input and generates binary file to be uploaded into Arduino.

Assembling blocks is done using a visual program editor inside IDE. To create a program, a programmer have to drag a block from toolbox and connects it to another block. Main use of the program editor is to manipulate blocks into a program via drag and dropping. User may save their program and load it back when needed. Generated source code are also shown and updated in real time, so whenever user modify the program, the changes will be reflected instantly in the generated code. User also may export the generated code for further editing in Arduino IDE. A simple serial monitor also included in the editor. It can send and receive serial message from any serial port given with adjustable baud rate.

Transformation from code in block form to Arduino language is done in real time, as the block is placed. Each block has their own mapping and similarly named to equivalent code in Arduino language. For example, to write a digital value to a pin function digitalWrite(pin_number, digital_value) is used. Equivalent block in Arduviz is named digitalWrite with two slots for block, each labelled pin and value respectively. This way, programmer can see the correlation between code in block form and in Arduino language easier than non real time code generation because the new segment of code is shown at the same time as the block is placed or moved. Real time code generation is achieved by making the block dispatch event when moved or placed. By listening to the event, the editor will reevaluate the block to generate new appropriate code.

Compiler used in this environment is avr-gcc. Before compiling the code, it needs to be transformed from Arduino language to C++ using an open source tool called Arduino Builder (https://github.com/arduino/arduino-builder). Binary file generated from compiler then uploaded to the Arduino using open source uploader, AVRDUDE (http://www.nongnu.org/avrdude/). Program editor have interface to access both of compiler and uploader, and can display output messages from those tools. Editor, compiler, and uploader are bundled into one environment, so there are no external dependencies needed.

A. Development Process

Development of Arduviz can be divided into three process: design, implementation, and testing. Design process of Arduviz involves designing the visual language and its mapping to Arduino language, choosing appropriate external AVR compiler and uploader, and designing the program editor. A prototype is made in implementation process. Framework and library needed to implement the IDE is determined in this process. Most of development time is spent on implementing the IDE. That prototype is then tested using test cases in testing process.

B. Visual Programming Language Design of Arduviz

There are three types of visual programming language: node-based, graphical rewrite rules, and block-based [6]. User can make program by stacking blocks vertically using this approach. Block-based visual programming can minimize mistake by limiting connectivity between blocks. One downside of this approach is, as the program grows, the blocks take lot of spaces. All example of existing visual programming tool mentioned before also use this approach.

To help people relate the visual language with the Arduino language, the visual programming language have to be easily related to the Arduino language counterpart. So, almost all of the blocks have same name as equivalent function in Arduino language. Each block also have tooltip when hovered, which give user information about that block.

Overall, the blocks is divided into two categories; value block and statement block. Value block is a block which returns value. It is used for variables, constants, and functions that return value. Statement block in other side doesn't return value, but can be stacked vertically to form a program. Beside of those two categories, there are also special blocks that cannot be attached to another blocks and can only contain statement blocks. It's used for setup() and loop() procedures, and anything outside those procedure such as function or variable declaration.

Blocks also can be divided into three main categories; structure, values, and functions. Structure contains conditionals such as if..else block and switch, loops such as while and for, and various operators. Values category handles data type, constants used in Arduino language and variable declarations. Functions category contains various predefined functions in Arduino such as digitalWrite() and digitalRead() for digital I/O. Arduino feature support of Arduviz is slightly less complete than Ardublock and more than miniBloq. miniBloq lacks proper support for programming serial communication whereas Arduviz can handle that just fine. Arduviz also support many helper functions in Arduino language, which miniBloq doesn't.

A block consists from various element, depending on the type. Common element of all block types is the body. Body of the block have text that indicates block name. Except special blocks mentioned before, all blocks have some sort of connectivity to other adjacent blocks. Value blocks have connectivity tab in the left side, indicates value returned by the block, while Statement block have connectivity tab in top and bottom of the block, enables it to be stacked one after another, forming a program. Difference between value block, statement block, and special block can be seen from illustration in Fig 1, Fig 2, and Fig 3 respectively.

Each block can also have text fields, value blocks or statement blocks as input. Common use case for a block with text field input is when we need a predetermined input value. User can enter value in the text field then the block will return that value. Example of block with text field input can be seen in Fig 4. Value block input is used for many purpose, such as assignment to a variable and function arguments. Illustration of

block with value input can also be seen in Fig 1. Statement input is used for blocks that can contain another statement, such as function declaration and control statements. Illustration for such block can also be seen in Fig 3.

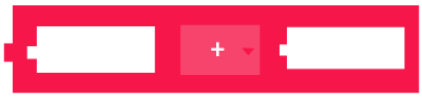

Fig 1. Example of value block

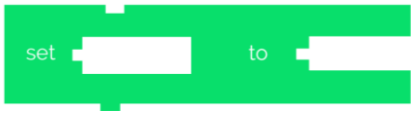

Fig 2. Example of statement block

Fig 3. Example of special block

Fig 4. Block with text input

Checking type of data is done simultaneously when assembling block. Each block has output type defined. Each block input also have that type information to ensure only correct block can be assigned to that input.

Block visual appearance in Arduviz is similar with Ardublock. Main difference between Arduviz and Ardublock in term of block design is the naming and mapping between syntax in arduino language and the block itself. Ardublock uses a layer of abstraction by using easily understandable block name, such as "Set digital output" to set digital value of a pin. On the other hand, Arduviz uses exact same function name as the equivalent function in Arduino language. This way, correlating block in Arduviz with equivalent syntax in Arduino language is easier.

Translation from block to Arduino code is done from most inside blocks first. For example, if we have an "if" statement, statements inside that block will be evaluated first into a code snippet in Arduino language. Then, that snippet is combined with if syntax, forming a proper if statement. This also applies to the input value blocks in a statement block. The value blocks are processed first to get the translated code of those blocks, then it's used to form a proper statement

C. *Implementation*

Prototype of this concept has been made using Blockly, a Javascript library for building visual programming editor. Blockly was chosen because it's open source and have extensive documentation to help development process. It use interlocking block to represent code concept. Blockly provides easy way to define custom blocks with various parameters such as block name, connectivity, and input. It also allows blocks to be easily translated to textual source code.

Blockly is a Javascript library designed to be run on the browser. However, we need to access file system and call compiler and uploader as external program. Therefore Electron is used to enable that. Electron is a framework that allows development of native desktop application using web technologies. It uses NodeJS runtime for the backend and Chromium to render page. Since Blockly and Electron was used, the implementation is done mostly in Javascript and HTML with some CSS styling.

Program editor consists of four main area depicted in Fig 8. Area A is the main workspace area where the programmer manipulate blocks by drag and dropping. The left side inside Area A is occupied with a toolbar that contains blocks to be used. Block that has been assembled can be deleted by dragging that block to recycle bin icon in bottom right of the editor area. Each block also have context menu when right-clicked, showing extra actions. When hovered, a block will show additional information regarding that block. Area B is used to preview generated source code. The source code is generated when the block is placed in real-time. It then can be exported to a sketch file and can be opened in Arduino IDE. Area C is used to display various output, either from the IDE or from the compiler and uploader. Area D is used to send and display serial message from a connected Arduino board.

This IDE is bundled with avr-gcc and avrdude as compiler and uploader respectively. The reason for that is to simplify installation process, as one can simply copy the files and run the provided executable to use the IDE. Also, it ensures the compiler is compatible with the IDE.

D. *Tests*

To test the implementation, three sample project is made using visual programming environment that has been made. Those projects have different complexity, ranging from simple digital I/O to serial communication.

1) *Blink*

Blink project contains an LED lamp that will blink endlessly with a certain delay. This project test digital output feature in Arduino. The program is shown in Fig 5. Generated code can be seen in Listing 1.

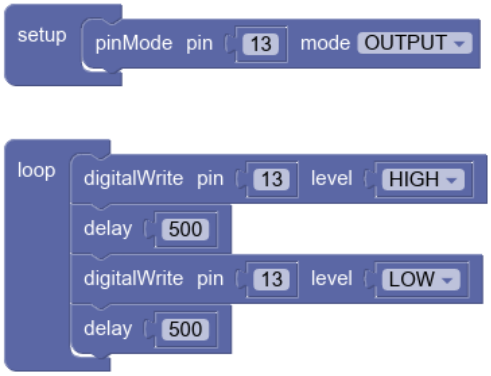

Fig 5. Blink program

```
void setup() {
  pinMode(13,OUTPUT);
}
```

```
void loop() {
  digitalWrite(13,HIGH);
  delay(500);
  digitalWrite(13,LOW);
  delay(500);
}
```

Listing 1. Generated code of blink program

2) Light sensor

Light sensor project contains a light dependent resistor (LDR) as an input to the system. Then, an LED lamp will be used to indicate whether the LDR is receiving light or not. This project test analog input feature in arduino. The program is shown in Fig 6. Generated code from this program can be seen in Listing 2.

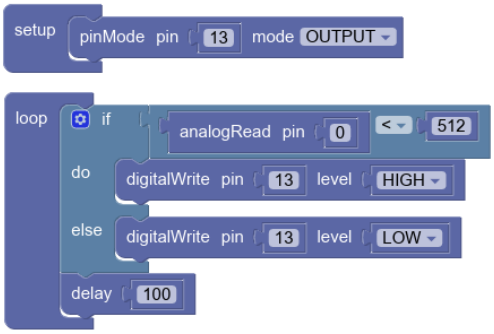

Fig 6. Light sensor program

```
void setup() {
  pinMode(13,OUTPUT);
}

void loop() {
  if (analogRead(0) < 512) {
    digitalWrite(13,HIGH);
  } else {
    digitalWrite(13,LOW);
  }
  delay(100);
}
```

Listing 2. Generated code of light sensor program

3) Simple Echo Server

Simple server project utilizes serial communication to a PC. The way it work is by sending back to the PC appropriate exact same message the arduino received from the PC. The program is shown in Fig 7. The generated code can be seen in Listing 3.

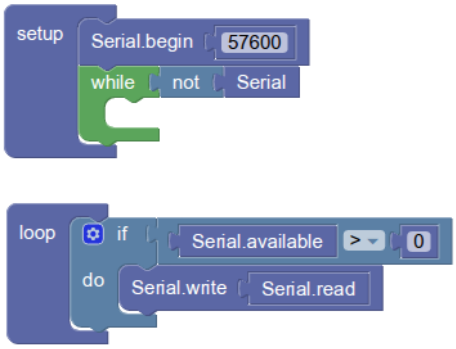

Fig 7. Simple echo server program

```
void setup() {
  Serial.begin(57600);
  while (!Serial) {
  }
}

void loop() {
  if (Serial.available() > 0) {
    Serial.write(Serial.read());
  }
}
```

Listing 3. Generated code of echo server program

V. Conclusion

By combining strength of Ardublock and miniBloq into an IDE, we can use most of those strength, such as real time code generation and support for many Arduino features to create a program. Also, setup process is simplified by bundling all needed tools into one development environment. By doing that, there's no external dependencies needed.

There are some aspect of this IDE that can be improved. The user interface and user experience could be improved to help people use the IDE more easily. Syntax highlighter could be added to make generated code easier to read and understand. Also, support for more Arduino features such as I^2C and SPI is nice to have.

Acknowledgment

We would like to thank Dr.Ir. M.M. Inggriani for encouraging us to submit this research to ICoDSE and her great assistance in this research by guiding and reviewing our writing.

References

[1] Banzi, M. and Shiloh, M, *Make: Getting Started with Arduino*. California: Maker Media, Inc, 2015.

[2] Arduino, "Arduino – Getting Started," January 2017. [Online]. Available: https://www.arduino.cc/en/Guide/HomePage. [Accessed: June. 29, 2017]

[3] Arduino, "Arduino – Sketch," 2017. [Online]. Available: https://www.arduino.cc/en/Tutorial/Sketch. [Accessed: June. 28, 2017]

[4] Arduino, "Arduino – Build Process," 2017. [Online]. Available: https://www.arduino.cc/en/Hacking/BuildProcess. [Accessed: June. 29, 2017]

[5] P. T. Cox, "Visual Programming Languages," *Wiley Encyclopedia of Computer Science and Engineering*, 2008, pp. 1 – 10.

[6] Roque, Vallarta Ricarose, "An Extendable Framework for Graphical Block Programming Systems," 2007. Massachusetts Institute of Technology

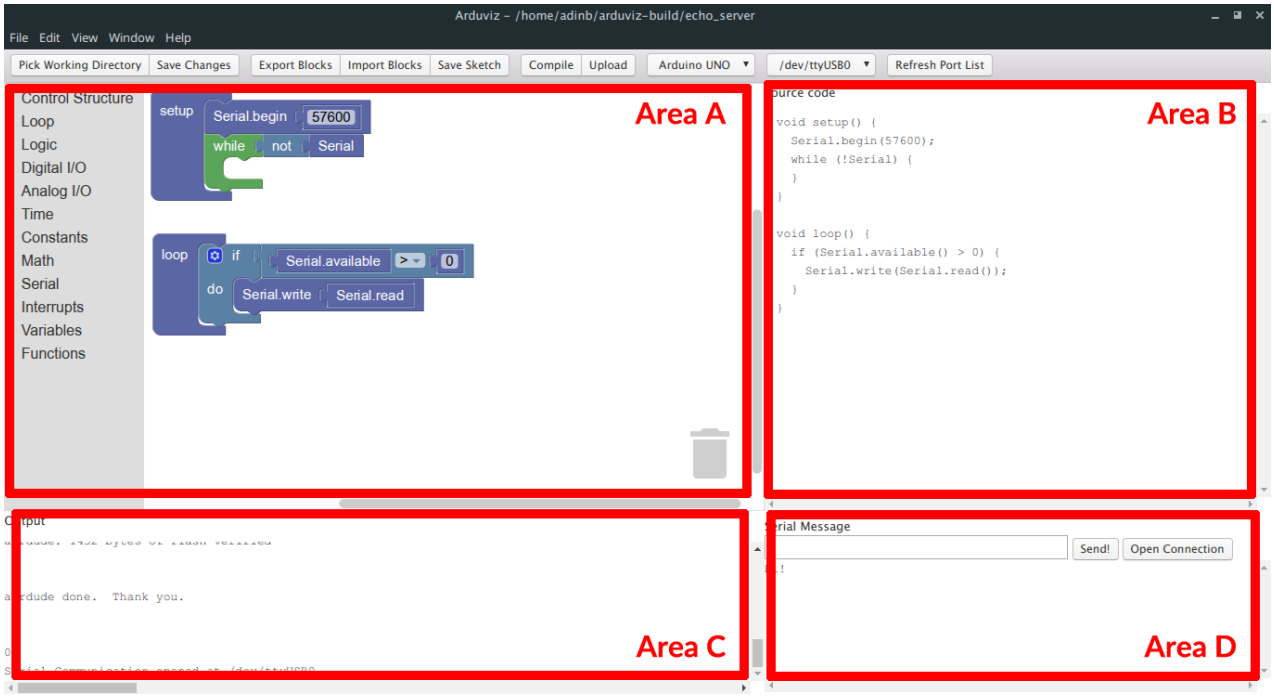

Fig 8. Interface of the IDE