A WEB-BASED DOMAIN SPECIFIC COMPILER FOR DESIGNING EMBEDDED SYSTEMS

C. W. S. Goonetilleke[1] and C. J. Basnayakege[2]**

[1,2]*Department of Electrical and Computer Engineering, The Open University of Sri Lanka*

INTRODUCTION

A Domain-Specific Language (DSL) is a computer language which specializes to a particular application domain. The realization of DSL differs in fundamental ways from that of traditional programming languages or General-Purpose Language (GPL). In contrast to a GPL, which is broadly applicable across domains, DSL lacks specialized features for a particular domain. DSL can be further subdivided by the kind of language, including domain-specific markup languages, domain-specific modeling languages, and domain-specific programming languages (Mernik, Heering, & Sloane, 2005). Domain Specific Compilers (DSCs) are used to compile a DSL and generate byte code or intermediate GPL code from DSL (Spinellis, 2001). Since DSLs allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain (Mernik, Heering, & Sloane, 2005). The idea is, domain experts (in this case 8051 based developers) may understand, validate, modify, and often even develop DSL programs. But it is involved with the cost of learning a new language vs its limited applicability. Also the difficulty of balancing trade-offs between domain-specificity and GPL constructs like C language and Assembly Language Program (ALP) (Gregor, Stanislav, & Giovanni, 2010).

To overcome the above difficulties involved with DSL, DSCs can be designed as Visual Programming Language (VPL) which lets users create programs by manipulating program elements graphically rather than by specifying them textually (Enderle, Guenther, Hilscher, & Kenn, 2009). Scratch is a famous example for VPL. In Scratch there is no compilation step or edit/run mode distinction. Users can click on a command or program fragment at any time to see what it does (Gregor, Stanislav, & Giovanni, 2010). But in hardware based VPL design, logic flow is specifically depended on the hardware integration and therefore it is needed to have VPL code compilation rather than instant interpretation. Unlike Scratch Flow code is an example for a commercially available VPL which can be used to program embedded devices such as PIC, AVR and ARM using flowcharts instead of a textual programming language and complies the VPL code against hardware design. But with VPLs, it is needed to create additional modules to manipulate new hardware and it also involves with lacking optimizations to binary code (Enderle, Guenther, Hilscher, & Kenn, 2009). Also, when comparing to textual based DSL, VPL based DSL is not suitable to implement complex codes in an efficient way since it keeps away developer further from the hardware level implementation of microcontroller (Gregor, Stanislav, & Giovanni, 2010). Therefore, present VPLs are not suitable to get understand working principles and architecture of an embedded system or a microcontroller. Programmers might need to have sophisticated hardware equipment and software tools for programming using present DSL based VPL compilers (Enderle, Guenther, Hilscher, & Kenn, 2009).

By getting advantages of both DSL and VPL, DSL can be extended with VPL and the entire compilation tasks can be moved to separate server by enabling web access to VPL tools. This approach is titled as a Web-based Domain Specific Compiler (WDSC). Since WDSC is web based, the programmer can create their programs online via using internet connected devices. Ultimately the programmer can design and program microcontroller based embedded system with sensors and actuators using WDSC, which consists with drag and drop UI. The entire process-flow can be designed and programmed using simple flow diagrams. The programmer

Corresponding author: Email -* ***charitha.ws@gmail.com/cjbas@ou.ac.lk*

will be able to design an entire embedded system within a short duration and compile it via online by using WDSC. The programmer only has to download the generated byte code from WDSC into the microcontroller with a console cable and IC burner. The main objective of this project is to design the WDSC that provides these advantages by minimizing the above difficulties.

METHODOLOGY

WDSC consists of multiple levels of language implementation as shown in Figure 1. It has high level DSL generation from VPL, DSL to Assembly Language Program (ALP) compilation and byte code generation from the ALP.

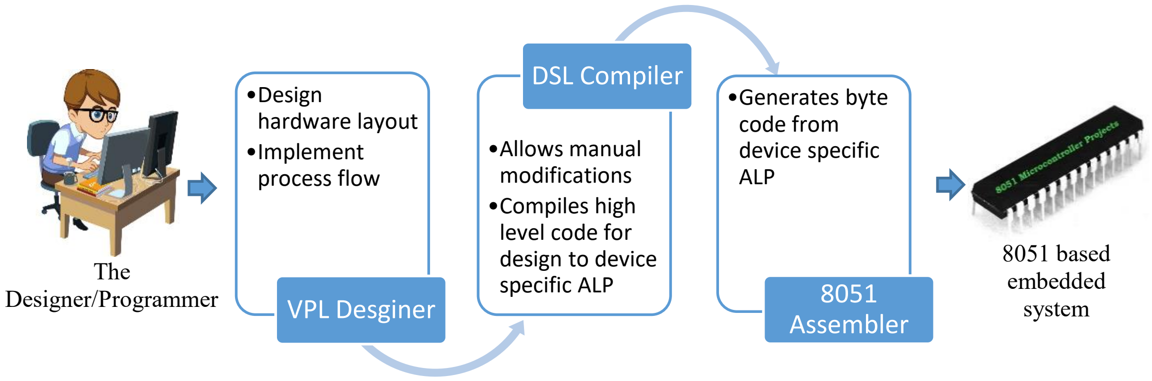

The Designer/Programmer
•Design hardware layout
•Implement process flow
VPL Desginer
DSL Compiler
•Allows manual modifications
•Compiles high level code for design to device specific ALP
•Generates byte code from device specific ALP
8051 Assembler
8051 Microcontroller Projects
8051 based embedded system

Figure 1. – Layers of WDS compiler

Figure 2 shows the WDSC framework of 8051 based embedded system application design. The VPL implemented as a web user interface (WUI). It has two designing stages. The stage one is to design a hardware connectivity (physical design) and the stage two is to design and implement a logical design respectively. Based on physical design and logical design, the VPL generates DSL code using Java script based VPL to DSL interpreter. Programmer can modify DSL manually if required using the Code View UI. Also the programmer can generate the ALP or can download the byte code as a HEX file by using the Code View. DSL compilation is done by the server side DSC. The DSL is uploaded to the server for compilation from the client side. DSC is implemented using C++ and with the help of Lex and Yacc tools for implementing the lexical analyzer and the syntax analyzer. The code generator for DSC is also written by using C++. Server side also consists with 8051 macro assembler to generate the HEX file from the ALP.

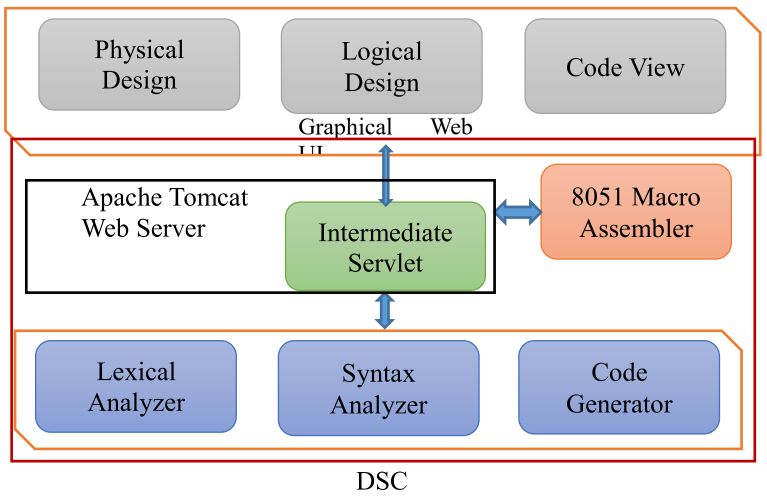


Figure 2. – WDSC framework

GRAMMAR BEHIND THE DSC

There are four types of grammars available in deriving the DSC (Krithivasan, 2015). Basically DSC uses context-free grammar. The Grammar *G* of the DSC is shown in the Figure 3.

N = { FUNCTION, STATEMENT, CONDITION, EXPRESSION, OPERATION, BITEXPRESSION, LOGIC }

T = { end, while, if, else, clk, '(', ')', '{', '}', ';', '.', '=', '+', '-', '', '/', '<', '>', '>=', '<=', '!=', '==', delay, [1-9][0-9]*, P[0-3], R[0-7], B[0-7][A-F0-9], [0-1]b }*

S → FUNCTION 'end'

FUNCTION → FUNCTION STATEMENT | STATEMENT

| FUNCTION while (CONDITION) {FUNCTION}

| FUNCTION if (CONDITION) {FUNCTION}

| FUNCTION if (CONDITION) {FUNCTION} else {FUNCTION}

STATEMENT → clk = [1-9][0-9]; | delay [1-9][0-9]*; | P[0-3] = EXPRESSION ;*

| R[0-7] = EXPRESSION ; | R[0-7] = EXPRESSION OPERATION EXPRESSION ;

| P[0-3]"."[0-7] = BITEXPRESSION; | B[0-7][A-F0-9] = BITEXPRESSION ;

CONDITION → EXPRESSION LOGIC EXPRESSION | BITEXPRESSION LOGIC BITEXPRESSION

*EXPRESSION → R[0-7] | P[0-3] | [1-9][0-9]**

BITEXPRESSION → B[0-7][A-F0-9] | P[0-3]"."[0-7] | [0-1]b

OPERATION → '+' | '-' | '' | '/'*

LOGIC → '<' | '>' | '>=' | '<=' | '!=' | '=='

Figure 3. – Grammar of the DSC

G = {N, T, P, S}; Where, N – Nonterminal strings, T – Terminal strings, P – Production rules and S – Start symbol

RESULTS

To verify the WDSC functionality sample scenario carried out by implementing digital weight scale. Figure 4(a) showing connected weight sensor and Seven segment display while Figure 4(b) showing the logic flow and code snippets. Figure 5 showing the DSL code and generated ALP.

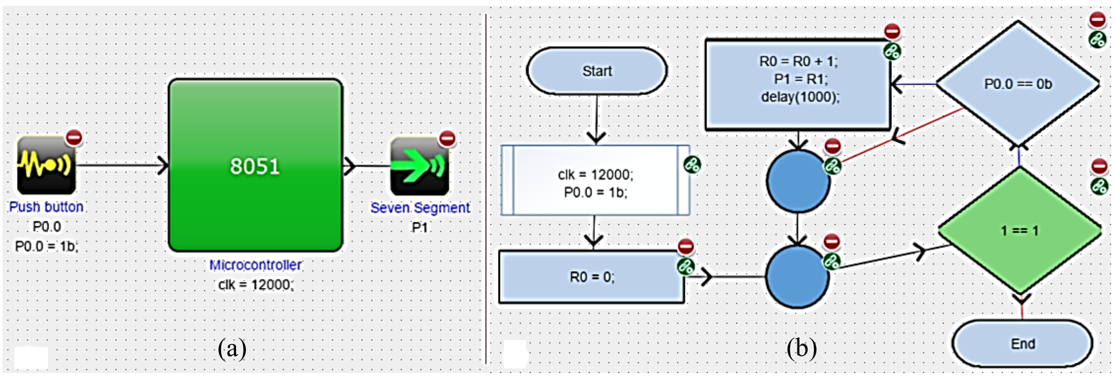
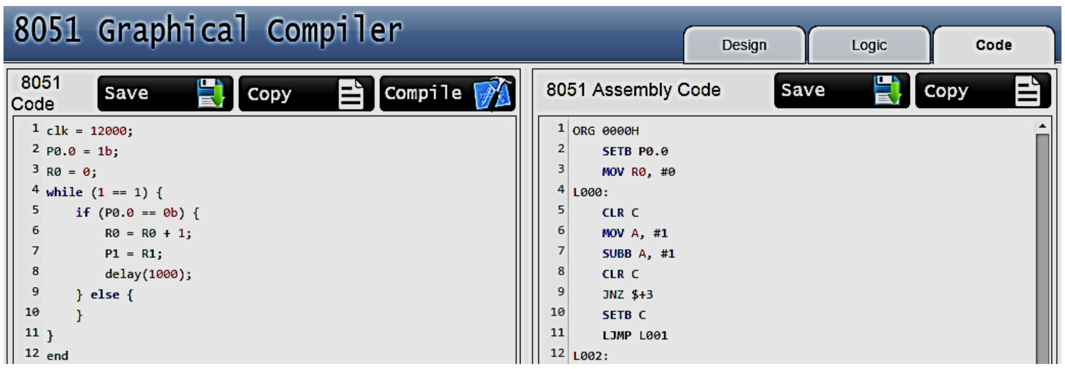


Figure 4. – (a) Physical Design & (b) Logical Design WUIs



```
clk = 12000;
P0.0 = 1b;
R0 = 0;
while (1 == 1) {
    if (P0.0 == 0b) {
        R0 = R0 + 1;
        P1 = R1;
        delay(1000);
    } else {
    }
}
end
```

```
ORG 0000H
    SETB P0.0
    MOV R0, #0
L000:
    CLR C
    MOV A, #1
    SUBB A, #1
    CLR C
    JNZ $+3
    SETB C
    LJMP L001
L002:
```

Figure 5. – Code View with DSL program and ALP

CONCLUSIONS

The WDSC accepts generic forms of expressions and is not hard coded for specific task. Since it graphically allows to map any sensor input in both bits and byte formats, various sensors and actuators can be incorporated as the digital weight scalar example. The logical design WUIs provide necessary code-blocks and logical conditions to work on different domains of applications. Built in delays, while loop and if, else conditions provide much more capability to the WDSC. Since the WDSC is integrated with both VPL and DSL, it provides more user friendly programming environment.

REFERENCES

Enderle, S., Guenther, W., Hilscher, H., & Kenn, H. (2009). xROB-S and iCon-X: Flexible Hardware, Visual Programming and Software Component Reuse. In *RoboCup 2008: Robot Soccer World Cup XII* (pp. 485--494). Berlin: Springer-Verlag.

Gregor, K., Stanislav, S., & Giovanni, G. (2010, June). Domain specific model-based development of software for programmable logic controllers. *Computers in Industry*, pp. 419-431.

Krithivasan, K. (2015, May 1). *Theory of Automata, Formal Languages and Computation*. Retrieved May 1, 2015, from NPTEL : Computer Science and Engineering: http://nptel.ac.in/courses/index.php?subjectId=106106049

Mernik, M., Heering, J., & Sloane, A. (2005). When and How to Develop Domain-specific Languages. *ACM Computing Surveys*, 316--344.

Solutions, M. T. (2015, May 1). *Matrix - Flowcode - a graphical programming language*. Retrieved from Flowcode - Electronic system design software: http://www.matrixtsl.com/flowcode/

Spinellis, D. (2001). Notable Design Patterns for Domain-specific Languages. *Journal of Systems and Software*, 91--99.

ACKNOWLEDGEMENTS

We are thankful to our colleges who have provided their valuable feedback and comments upon our research and who have spent their time with us to test and evaluate research outcomes.