Software UART: A Use Case for VSCPU Worst-Case Execution Time Analyzer

Abdullah Yıldız*, Deniz İskender‡, Gülçe Özlü*, H. Fatih Ugurdag†, Barış Aktemur‡, and Sezer Gören*
*Dept. of Computer Engineering, Yeditepe University, Istanbul, Turkey
Email: ayildiz@cse.yeditepe.edu.tr, gulce.ozlu@std.yeditepe.edu.tr, sgoren@cse.yeditepe.edu.tr
†Dept. of Electrical and Electronics Engineering, Özyeğin University, Istanbul, Turkey
Email: fatih.ugurdag@ozyegin.edu.tr
‡Dept. of Computer Engineering, Özyeğin University, Istanbul, Turkey
Email: deniz.iskender@ozu.edu.tr, baris.aktemur@ozyegin.edu.tr

***Abstract*—This paper presents our early results of the development of a Worst-Case Execution Time (WCET) analyzer for VSCPU by implementing a software UART system. Our WCET analyzer takes a C program as input and gives the time taken by each function as output. A software UART system eliminates the need to employ a dedicated hardware for RS232 interface and makes directly use of the processor instead. For this purpose, we designed and implemented a memory-mapped system which has access to UART pins and is capable of setting and sampling these pins from software by a method of bit banging. We used the output of our WCET analyzer to approximate the actual bit times for a specific UART baud rate. By successfully testing and verifying our software UART, we showed that our WCET analyzer could be used to estimate runtime of tasks in an application. Although development of the WCET analyzer is ongoing, the results are promising.**

***keywords*—WCET, UART, CPU, bit banging, FPGA, processor**

I. Introduction

One of the most critical characteristics of programs which are written for embedded and real-time systems is to check the program's compliance with time constraints [1]. The Worst Case Execution Time (WCET) of a program to be run is calculated to ensure that the time criteria are met. The vital importance of WCET analysis for real-time programs has necessitated the development of compilers' awareness of this concept [2]. For example, when compilers normally compile to optimize the average runtime of a program, real-time program compilers need to optimize for the longest harness path [3]–[5].

The calculated or measured WCET value should ideally be both reliable (i.e. it can be assured that there will be no runtime higher than the calculated value), and should be tight (i.e. as close to the actual WCET as possible). Dynamic methods based on measuring and executing the program can provide stringent values, but finding the input to ensure that the longest runway that is executed is generally insufficient for reliability. Static methods that make WCET estimation by analyzing the program have difficulty in providing strict enough upper limits while providing reliable inferences. This is due to the conservative nature of static methods, which must take into account inaccessible harness paths and the complexity of pipelines and cache structures in the processors.

In this paper, we present a static WCET analyzer for VSCPU. VSCPU [6] is a simple, customizable, educational, and easily implementable CPU with a complete toolchain. We show how to leverage our WCET analyzer to implement a software UART on VSCPU as a proof of concept.

This work is supported by TÜBİTAK under contract 117E090.

The paper is organized as follows. Section II introduces our proposed VSCPU WCET analyzer in detail. In Section III, we explain how memory-mapped VSCPU is designed and used to implement software UART application by bit banging. Section IV explains how VSCPU WCET analyzer is used to estimate bit time of a UART frame in software UART application. Section V shows our findings that includes how good our WCET analyzer performs to meet timing constraints of software UART application. Section VI concludes the paper.

II. VSCPU WCET Analyzer

VSCPU WCET analyzer is based on static program analysis in which properties of dynamic behavior of a given task are determined without actually executing the task [7]. VSCPU WCET analyzer is written as a Clang frontend action [8]. Given a C source file, the program is preprocessed, parsed, and semantically analyzed using Clang. Syntax and semantic errors (e.g. type mismatches) are detected, and Clang's high-quality error/warning messages are displayed. If no errors are detected, the abstract syntax tree (AST) of the input program is gathered. Then, WCET analyzer visits all the function declarations from the AST to collect the root functions creating AST-based call graph. If there is no cycle in the call graph, WCET analyzer performs a recursive depth-first traversal starting from the root node of call graph gathering the VSCPU assembly instructions for each functions. Each function is discovered with all possible execution paths with control flow analysis technique. There are three main categories of calculation methods proposed in literature for these possible execution paths: path, tree, or IPET (Implicit Path Enumeration Technique).

In the path-based bound calculation, the upper bound for a task is determined by computing bounds for different paths in the task, searching for the overall path with the longest execution time [9]. In this paper, path-based method is used. VSCPU compiler [6] that was already developed is supportive for WCET analysis during calculation phase. VSCPU compiler knows occurrence count of VSCPU instructions in any control flow node so that WCET of any node can be calculated by multiplying the occurrence count of VSCPU instructions and their cycles per instruction (CPI) counts (Table I), then summing up these cycle counts. The main difference of this analysis with literature is the target CPU called as VSCPU and its instruction list (Table I).

TABLE I
VSCPU CPI COUNTS

Instruction	Cycles per Instruction	Instruction	Cycles per Instruction
ADD	4	MUL	4
ADDi	3	MULi	3
ADDF	16	MULF	12
NAND	4	CP	3
NANDi	3	CPi	2
SRL	4	CPI	4
SRLi	3	CPIi/CPIr	4
LT	4	BZJ/BZ	4
LTi	3	BZJi/JMP	3
LTF	6		

```
for(int i = CONSTANT; i <= CONSTANT; i++) { }
for(int i = CONSTANT; i <= CONSTANT; ++i) { }
for(int i = CONSTANT; i >= CONSTANT; i--) { }
for(int i = CONSTANT; i >= CONSTANT; i--) { }
```

Listing 1. Supported for statements in our WCET analyzer

The WCET analyzer is able to handle array operations, functions with void and non-void return types, specific for-loops. The WCET analyzer has certain limitations as well. Currently, there is no support for structs and the switch statement. While loop and direct or indirect recursion are not supported so that the set of paths is always finite since termination must be guaranteed. The supported for statements are provided in Listing 1 and the assumption is that the variable that is inside in init, conditional, and incremented parts of the for statements is not changed in the body.

In the possible input to WCET analyzer shown in Listing 2, there are 4 functions. Each function has its own body which includes some function calls, if-else statement and for loop. Since init and conditional parts of for statement include constant values, for loop can be evaluated by WCET analyzer. Otherwise, the error messages are displayed to inform the user about init, conditional, or increment parts of the for statement. Output created by analysis in Listing 3 shows these 4 functions, how their WCET is calculated step by step and their WCET in terms of cycle.

```
int changeSign(int number) {
  return -number;
}

int add(int a, int b) {
  return (a + b);
}

int subtract(int z, int w) {
  return add(z, changeSign(w));
}

int timesTen(int x) {
  int result = 0;
  if (x < 0) {
    result = x * 10;
  } else {
    for (int i = 1; i <= 10; i++) {
      result += x;
    }
  }
  return result;
}

int main () {
  int num1 = 5;
  int num2 = -3;

  return(add(num1, num2) + subtract(num1, num2) + timesTen(
       num1) + timesTen(num2));
}
```

Listing 2. An example C Program for our WCET analyzer

```
------------------EACH FUNCTION'S WORST CASE EXECUTION TIME
      IN TERMS OF CYCLE------------------

FUNCTION NAME: changeSign TOTAL CYCLE: 73
------------------ END OF A FUNCTION------------------

FUNCTION NAME: add TOTAL CYCLE: 98
------------------ END OF A FUNCTION------------------

subtract : 127 add : 98
subtract's current total after calling function add: 225

subtract : 225 changeSign : 73
subtract's current total after calling function changeSign:
      298

FUNCTION NAME: subtract TOTAL CYCLE: 298
------------------ END OF A FUNCTION------------------

FUNCTION NAME: timesTen TOTAL CYCLE: 2149
------------------ END OF A FUNCTION------------------

main : 373 add : 98
main's current total after calling function add: 471

main : 471 subtract : 298
main's current total after calling function subtract: 769

main : 769 timesTen : 2149
main's current total after calling function timesTen: 2918

main : 2918 timesTen : 2149
main's current total after calling function timesTen: 5067

FUNCTION NAME: main TOTAL CYCLE: 5067
------------------ END OF A FUNCTION------------------
```

Listing 3. Example output generated by our WCET analyzer

To analyze if-else statement, if and else blocks are analyzed separately, the execution time of each block is compared with each other, then maximum one is added to execution time of function. The formula is max(T(then), T(else)). During the depth first traversal, if there is any function call from the caller, callee becomes the new root to be traversed. If there is any function call from this callee, then the execution time of functions that are called from the this callee are added to execution time of the callee. Finally, execution time of the callee is added to the execution time caller function. This is how to recursively find the execution time of any function. Since main is also function, worst case execution time of the program is equal to the worst case execution time of the main function.

III. DESIGN AND IMPLEMENTATION OF SOFTWARE UART WITH VSCPU

UART protocol is an asynchronous serial communication mechanism in which there is no dedicated clock signal between each communication end. Figure 1 shows the timing sequence of an example UART frame which includes 8 data bits, 1 stop bit, and no parity bit. The time which passes between each consecutive number corresponds to reciprocal of baud rate which could be one of the standard UART baud values (e.g., 9600, 19200, 115200, etc.). For example, the bit time for 115200 baud corresponds to about 8680 ns.

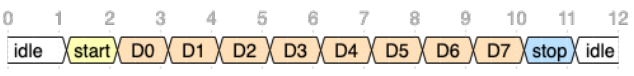

Fig. 1. Example timing sequence of a UART frame (8N1)

Normally, there is a dedicated UART hardware on each communication end which handles signalling and transferring data between them without direct intervention of processor.

```
...
7: CPi 4096 0 // send the start bit
8: CP 56 81 // loop head for start bit
9: ADD 56 511
10: CP 57 56
11: LTi 57 1
12: BZJ 84 57 // loop end for start bit
...
18: CP 4096 62 // send the next bit
...
21: CP 56 80 // loop head for data bit
22: ADD 56 511
23: CP 57 56
24: LTi 57 1
25: BZJ 85 57 // loop end for data bit
...
```

Listing 4. Driving UART TX pin from software with bit banging

Here, we completely remove such a hardware and make use of the processor. In order to generate or provide the respective bit time for each bit either to be received or transmitted, a delay loop inside the program could be used. If we can repeat this loop for each bit, then we can send/receive data by the processor just as if using a dedicated UART hardware. This technique is called **bit banging**. Bit banging provides the required signaling of a communication protocol with a program which runs on the processor and sets the signal or samples it according to the timing and synchronization requirements of the communication protocol. Listing 4 shows a brief example on how one could send data over UART TX pin from software with bit banging for VSCPU [6].

As can be seen in Listing 4, instructions 7-12 provides sending the start bit of UART frame. The time which is spent on the loop (instructions 9-12) assures the correct bit time for the start bit. After sending the start bit, data bits and the stop bit are sent in a similar fashion over TX pin.

To implement software UART with VSCPU, memory-mapped I/O model technique [6] was employed to access RX and TX pins of UART peripheral from VSCPU. Figure 2 shows hardware block diagram of software UART design.

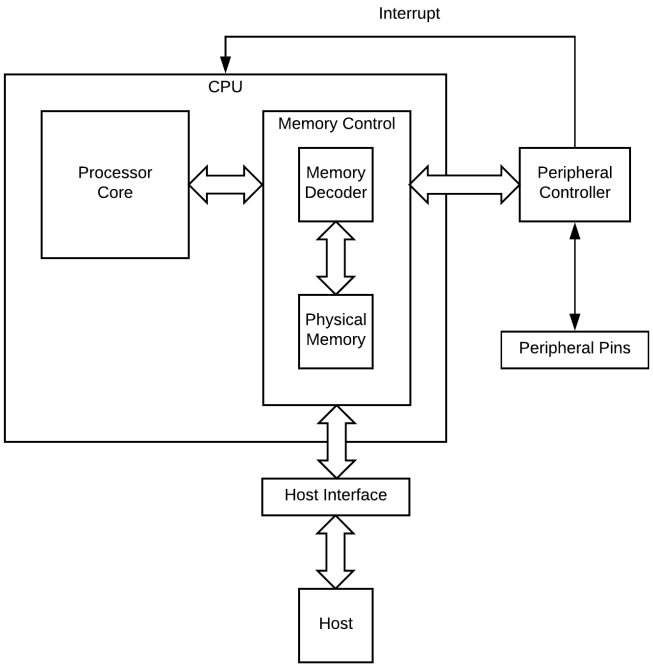

Fig. 2. Top-level hardware block diagram of software UART design [6]

Components that is shown in Figure 2 could be briefly explained as follows:

- ***Processor Core*** represents processor (i.e., VSCPU).
- ***Memory Decoder*** is responsible for transferring data between the processor and physical memory and peripherals based on the addresses which are referenced from the processor.
- ***Peripheral Controller*** includes control units for each peripheral including software UART.
- ***Host Interface*** provides downloading program onto physical memory and debugging of the processor.

The memory-mapped I/O model provides the processor to access both memory and peripherals from the same address space of the processor without extra instructions. Table II shows the memory map of VSCPU which is used to implement software UART.

TABLE II
MEMORY MAP OF SOFTWARE UART DESIGN

Section	Address Range
Local Memory	0x0000-0x0fff
Software UART	0x1000-0x100f
LED	0x1010-0x101f
VGA	0x1020-0x102f
Switch	0x1030-0x103f

As shown in Table II, there are five sections which are accessible by VSCPU within the memory map of this system. The first section (0x0000-0x0fff) is reserved for **Local Memory** which is the instruction and data memory (physical memory) of VSCPU. Notice that, the size of this section is 4096x32-bit. Software UART peripheral registers are in the address range of (0x1000-0x1010). Peripherals LED, VGA, and switch are used for debugging and testing of our design.

Memory map that is shown in Table II provides an addressable memory region of 16x32-bit for the software UART peripheral. Listing 5 shows Verilog HDL description of the software UART peripheral. Notice that the first bits of the first two registers are actively used. That is, **tx** pin of UART peripheral is connected to first bit of the first peripheral register and **rx** pin of UART peripheral is connected to first bit of the second peripheral register. Other bits and registers are reserved for future use.

According to the memory map that is shown in Table II, the first software UART register corresponds to address 0x1000 or decimal 4096. Hence, a VSCPU copy instruction such as **CPi 4096 1** sets the value in this register to one and hence VSCPU implicitly could send a logic 1 over UART TX pin. Similarly, the second software UART register corresponds to address 0x1001 or decimal 4097. Hence, a VSCPU copy instruction such as **CP 100 4097** reads the value in this register and hence VSCPU implicitly could sample the value which is sent over UART RX pin.

IV. WCET ANALYSIS OF SOFTWARE UART

As explained in Section II, our WCET analyzer can measure the time which is taken by a function. Therefore, we can make use of this feature of our WCET analyzer to approximate the bit time needed to transfer a bit in a UART frame. For this purpose, a C program which consists of respective functions to send start, data, and stop bits is required. Listing 6 shows an example C program for sending bytes over TX pin.

```
module software_uart_p(

  input clk,
  input rst,
  input wea,
  input [0:0] addra,
  input [31:0] dina,
  output reg [31:0] douta,
  output [0:0] tx,
  input [0:0] rx

  );

  reg [31:0] software_uart_p_r [0:1];

  always@(posedge clk) begin

    douta <= software_uart_p_r[addra];

    if(rst) begin
      software_uart_p_r[0] <= 0;
      software_uart_p_r[1] <= 0;
    end
    else if(wea) begin
      software_uart_p_r[addra] <= dina;
    end
    else begin
      software_uart_p_r[1][0] <= rx;
    end

  end

  assign tx = software_uart_p_r[0][0];

endmodule
```

Listing 5. HDL description of VSCPU Software UART Peripheral

```
void start_bit_delay(){
  for(i=0;i<115;i++){
  }
}

void data_bit_delay(){
  for(i=0;i<115;i++){
  }
}

void stop_bit_delay(){
  for(i=0;i<115;i++){
  }
}

void send_bit(int bit_value){
  *tx = bit_value;
}

void send_data_bit(int data_bit){
  send_bit(data_bit);
  data_bit_delay();
}

void send_start_bit(){
  send_bit(0);
  start_bit_delay();
}

void send_stop_bit(){
  send_bit(1);
  stop_bit_delay();
}

void send_byte(int data){
  for(j=0;j<8;j++){
    lsb = (data >> j) & 1;
    send_data_bit(lsb);
  }
}
```

Listing 6. Code snippet from a C program which sends bytes over TX pin

```
send_data_bit : 102 send_bit : 86
send_data_bit's current total after calling function
     send_bit: 188
send_data_bit : 188 data_bit_delay : 10238
send_data_bit's current total after calling function
     data_bit_delay: 10426
FUNCTION NAME: send_data_bit TOTAL CYCLE: 10426
------------------ END OF A FUNCTION------------------
send_start_bit : 92 send_bit : 86
send_start_bit's current total after calling function
     send_bit: 178
send_start_bit : 178 start_bit_delay : 10238
send_start_bit's current total after calling function
     start_bit_delay: 10416
FUNCTION NAME: send_start_bit TOTAL CYCLE: 10416
------------------ END OF A FUNCTION------------------
send_stop_bit : 92 send_bit : 86
send_stop_bit's current total after calling function
     send_bit: 178
send_stop_bit : 178 stop_bit_delay : 10238
send_stop_bit's current total after calling function
     stop_bit_delay: 10416
FUNCTION NAME: send_stop_bit TOTAL CYCLE: 10416
------------------ END OF A FUNCTION------------------
```

Listing 7. Output of our WCET analyzer for the C program in Listing 6

As shown in Listing 6, there is no specific directive or constraint to notify the compiler about the baud rate or the bit time. Instead, we use delay functions for each bit. The iteration count simply defines the time in which the value of the corresponding bit remains constant at TX pin. Hence, we can use the output of our WCET analyzer to approximate bit times of a UART frame. Listing 7 shows the corresponding output generated by our WCET analyzer for the C program in Listing 6.

The C program is written for transmitting bytes over TX pin at a baud rate of 9600. Hence, each bit time should be about 104160 ns. When we look at the Listing 7, we can see that the time taken in functions **send_data_bit**, **send_start_bit**, and **send_stop_bit** are 10426, 10416, and 10416 cycles, respectively. If we use a clock signal of 100 MHz, the time taken in these functions correspond to 104260, 104160, and 104160 ns, respectively. Hence, we can clearly say that these measurements nearly match the bit time of 104160 ns for 9600 baud. The only drawback in using our WCET analyzer is the need to find the correct iteration counts manually to approximate the bit times.

V. Tests and Results

Three standard baud rates (9600, 19200, 115200) are used to test and verify our WCET analyzed software UART design. We verified our proposed design by simulation on Xilinx ISE 14.7. We measured bit times for transferring data (TX) and sampling times for receiving data (RX) which are approximated by using our WCET analyzer.

We selected two different C programs for WCET analysis. The first one uses dedicated functions to send each bit in a UART frame as shown in Listing 6 while the second one encapsulates all statements related to bit banging and delay loops in a single function as shown in Listing 8. By doing so, we wanted to see the effect of function call and return operations on transferring and receiving data. In order to see how well our WCET analyzer performs, we also compared WCET analyzed approach with another one in which VSCPU assembly programs are used to transfer UART frames. In this approach, we wrote a Python script that generates corresponding VSCPU assembly program for a specific baud rate.

```
void send_data_array(int data){
  *tx=0;
  for(i=0;i<115;i++){}
  lsb = (data >> 0) & 1;
  *tx=lsb;
  for(i=0;i<115;i++){}
  lsb = (data >> 1) & 1;
  *tx=lsb;
  for(i=0;i<115;i++){}
  lsb = (data >> 2) & 1;
  *tx=lsb;
  for(i=0;i<115;i++){}
  lsb = (data >> 3) & 1;
  *tx=lsb;
  for(i=0;i<115;i++){}
  lsb = (data >> 4) & 1;
  *tx=lsb;
  for(i=0;i<115;i++){}
  lsb = (data >> 5) & 1;
  *tx=lsb;
  for(i=0;i<115;i++){}
  lsb = (data >> 6) & 1;
  *tx=lsb;
  for(i=0;i<115;i++){}
  lsb = (data >> 7) & 1;
  *tx=lsb;
  for(i=0;i<115;i++){}
  *tx=1;
  for(i=0;i<115;i++){}
}
```

Listing 8. Revised version of the C program in Listing 6

A. Timing Results

Table III shows error values in terms of nanoseconds along with their percent error which state how much measured bit times deviate from expected bit times for each approach in order to transmit a UART frame at 9600, 19200, and 115200 bauds, respectively. Results in question show that using assembly programs which are generated by the Python script performs very well and matches the expected bit time. However, both WCET analyzed C programs exhibit deviations from the expected bit time.

Table IV shows error values in terms of nanoseconds along with their percent error which state how measured sampling times deviate from expected sampling times for each approach in order to receive a UART frame at 9600, 19200, and 115200 bauds, respectively. Assembly programs which are generated by the Python script again performs very well and nearly matches the expected sampling times. As in sending the UART frame, results for WCET analyzed C programs exhibit deviations from the expected sampling time.

The large amount of error in case of using our WCET analyzer is due to its lack of knowledge about the cost of function call and return operations. Since we only take the cost of the function into account in order to find the approximate bit time, these function call and return operations add up to the function cost and hence cause bit/sampling time to deviate from the expected value. In this case, it is better to encapsulate all the operations in a single function (as shown in Listing 8) and then approximate the bit time for each bit by dividing the cost of **send_data_array** function (which was given our WCET analyzer) by 10 since there are 10 bits in a UART frame. As shown in both Table IV and IV, using dedicated functions in order to send each bit rather than using a single function to send each bit one after the another causes larger deviations in expected bit and sampling times.

As a result of that, depending on the baud rate, the additional cost which arises from function calls and returns could cause data loss in both transmitting and receiving data. We tested and noticed that these errors cause data loss in our system for 19200 and 115200 bauds when transmitting data and for 115200 baud when receiving data.

B. Area Results

We also evaluated our proposed WCET analyzed software UART design in terms of hardware area utilization. For this purpose, we implemented our system on Digilent NEXYS4-DDR board which features Xilinx Artix-7 XC7A100T. Table V shows respective slice utilizations for three different designs: the first one is ***HW UART*** which only includes a dedicated UART circuit. The second design is ***HW UART w/ VSCPU*** which is a memory-mapped system which includes VSCPU and a dedicated UART peripheral circuit. The third design ***SW UART w/ VSCPU*** is a memory-mapped system which includes VSCPU and a software UART peripheral circuit as explained in Section III.

According to the results in Table V, ***SW UART w/ VSCPU*** design is located somewhere between the two other designs. This is reasonable because ***HW UART*** design just consists of a dedicated UART circuit and ***HW UART w/ VSCPU*** design includes both VSCPU and a dedicated UART circuit which is more complex than the software UART circuit.

VI. Conclusion

In this paper, we presented our ongoing work on development of a WCET analyzer for VSCPU. Our proposed WCET analyzer is essentially a depth-first traversal algorithm which is based on method declarations and it generates each methods' WCET for a given C program. In addition to introducing a WCET analyzer for VSCPU, we designed and implemented software UART to show how this WCET analyzer could simplify a programmer's effort when writing such applications in a high-level language such as C. We then evaluated the performance of our WCET analyzed C programs and compared it with the performance of VSCPU assembly programs which are generated by a Python script.

Despite the accuracy of our WCET analyzer is promising, it is still under development and in the future, we plan to add features into it such as measuring execution time not only for functions but for specific parts of a program. Therefore, it will be possible to better estimate the time taken between two parts of a program and this will improve the accuracy of our WCET analyzer.

Acknowledgment

This work is supported by TÜBİTAK under contract 117E090. All documentation, hardware design files (RTL codes) and software source codes are available at https://github.com/MC2SC.

References

[1] P. Axer, R. Ernst, H. Falk, A. Girault, D. Grund, N. Guan, B. Jonsson, P. Marwedel, J. Reineke, C. Rochange, M. Sebastian, R. V. Hanxleden, R. Wilhelm, and W. Yi, "Building timing predictable embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 4, pp. 82:1–82:37, Mar. 2014. [Online]. Available: http://doi.acm.org/10.1145/2560033

[2] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann *et al.*, "T-CREST: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 449–471, 2015.

[3] M. M. Kafshdooz, M. Taram, S. Assadi, and A. Ejlali, "A compile-time optimization method for wcet reduction in real-time embedded systems through block formation," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, pp. 66:1–66:25, Jan. 2016. [Online]. Available: http://doi.acm.org/10.1145/2845083

TABLE III
ERROR VALUES IN NS AND PERCENT ERROR FOR EACH BIT IN A UART FRAME WITH RESPECT TO EXPECTED BIT TIME
(TX, 9600/19200/115200 BAUD, 8N1)

	9600 baud			19200 baud			115200 baud		
Method / Bits	Assembly Code Generator	WCET Analyzed C (Single Function)	WCET Analyzed C (Multiple Functions)	Assembly Code Generator	WCET Analyzed C (Single Function)	WCET Analyzed C (Multiple Functions)	Assembly Code Generator	WCET Analyzed C (Single Function)	WCET Analyzed C (Multiple Functions)
start	0 (0%)	+650 (+0.62%)	+3590 (+3.45%)	0 (0%)	+810 (+1.56%)	+3750 (+7.20%)	0 (0%)	+1090 (+12.56%)	+3150 (+36.29%)
D0	0 (0%)	-310 (-0.30%)	+2780 (+2.67%)	-10 (-0.02%)	-150 (-0.29%)	+2940 (+5.65%)	0 (0%)	+130 (+1.50%)	+2340 (+26.96%)
D1	0 (0%)	-310 (-0.30%)	+2780 (+2.67%)	-10 (-0.02%)	-150 (-0.29%)	+2940 (+5.65%)	0 (0%)	+130 (+1.50%)	+2340 (+26.96%)
D2	0 (0%)	-310 (-0.30%)	+2780 (+2.67%)	-10 (-0.02%)	-150 (-0.29%)	+2940 (+5.65%)	0 (0%)	+130 (+1.50%)	+2340 (+26.96%)
D3	0 (0%)	-310 (-0.30%)	+2780 (+2.67%)	-10 (-0.02%)	-150 (-0.29%)	+2940 (+5.65%)	0 (0%)	+130 (+1.50%)	+2340 (+26.96%)
D4	0 (0%)	-310 (-0.30%)	+2780 (+2.67%)	-10 (-0.02%)	-150 (-0.29%)	+2940 (+5.65%)	0 (0%)	+130 (+1.50%)	+2340 (+26.96%)
D5	0 (0%)	-310 (-0.30%)	+2780 (+2.67%)	-10 (-0.02%)	-150 (-0.29%)	+2940 (+5.65%)	0 (0%)	+130 (+1.50%)	+2340 (+26.96%)
D6	0 (0%)	-310 (-0.30%)	+2780 (+2.67%)	-10 (-0.02%)	-150 (-0.29%)	+2940 (+5.65%)	0 (0%)	+130 (+1.50%)	+2340 (+26.96%)
D7	0 (0%)	-1600 (-1.54%)	+1550 (+1.49%)	-10 (-0.02%)	-1440 (-2.76%)	+1710 (+3.28%)	0 (0%)	-1160 (-13.36%)	+1110 (+12.79%)
stop	-10 (-0.01%)	-1260 (-1.21%)	+1490 (+1.43%)	0 (0%)	-1100 (-2.11%)	+1650 (+3.17%)	-10 (-0.12%)	-820 (-9.45%)	+1050 (+12.10%)

TABLE IV
ERROR VALUES IN NS AND PERCENT ERROR FOR EACH BIT IN A UART FRAME WITH RESPECT TO EXPECTED SAMPLING TIME
(RX, 9600/19200/115200 BAUD, 8N1)

	9600 baud			19200 baud			115200 baud		
Method / Bits	Assembly Code Generator	WCET Analyzed C (Single Function)	WCET Analyzed C (Multiple Functions)	Assembly Code Generator	WCET Analyzed C (Single Function)	WCET Analyzed C (Multiple Functions)	Assembly Code Generator	WCET Analyzed C (Single Function)	WCET Analyzed C (Multiple Functions)
start	-10 (-0.02%)	-120 (-0.23%)	+170 (+0.33%)	-10 (-0.04%)	+400 (+1.54%)	+110 (+0.42%)	-10 (-0.23%)	+100 (+2.30%)	-190 (-4.38%)
D0	-20 (-0.01%)	-780 (-0.50%)	+200 (+0.13%)	-10 (-0.01%)	-1860 (-2.38%)	+300 (+0.38%)	-20 (-0.15%)	-2760 (-21.20%)	-600 (-4.61%)
D1	-20 (-0.01%)	-620 (-0.24%)	+2810 (+1.08%)	-20 (-0.02%)	-3300 (-2.53%)	+3070 (+2.36%)	-30 (-0.14%)	-4800 (-22.12%)	+1570 (+7.24%)
D2	-20 (-0.01%)	+60 (+0.02%)	+5420 (+1.49%)	-30 (-0.02%)	-4220 (-2.32%)	+5840 (+3.20%)	-40 (-0.13%)	-6320 (-20.80%)	+3740 (+12.31%)
D3	-20 (∼0%)	+740 (+0.16%)	+8030 (+1.71%)	-40 (-0.02%)	-5140 (-2.19%)	+8610 (+3.67%)	-50 (-0.13%)	-7840 (-20.07%)	+5910 (+15.13%)
D4	-20 (∼0%)	+1420 (+0.25%)	+10640 (+1.86%)	-50 (-0.02%)	-6060 (-2.12%)	+11380 (+3.97%)	-60 (-0.13%)	-9360 (-19.61%)	+8080 (+16.93%)
D5	-20 (∼0%)	+2100 (+0.31%)	+13250 (+1.96%)	-60 (-0.02%)	-6980 (-2.06%)	+14150 (+4.18%)	-70 (-0.12%)	-10880 (-19.28%)	+10250 (+18.17%)
D6	-20 (∼0%)	+2780 (+0.36%)	+15860 (+2.03%)	-70 (-0.02%)	-7900 (-2.02%)	+16920 (+4.33%)	-80 (-0.12%)	-12400 (-19.05%)	+12420 (+19.08%)
D7	-20 (∼0%)	+3460 (+0.39%)	+18470 (+2.09%)	-80 (-0.02%)	-8820 (-1.99%)	+19690 (+4.45%)	-90 (-0.12%)	-13920 (-18.87%)	+14590 (+19.78%)
stop	-30 (∼0%)	+4140 (+0.42%)	+21420 (+2.16%)	-80 (-0.02%)	-9740 (-1.97%)	+22800 (+4.61%)	-90 (-0.11%)	-15440 (-18.72%)	+17100 (+20.74%)

TABLE V
AREA UTILIZATION ON XILINX ARTIX-7 XC7A100T

Design / Resource	HW UART	HW UART w/ VSCPU	SW UART w/ VSCPU
# of Slice Registers	0.04%	0.22%	0.15%
# of Slice LUTs	0.18%	2.02%	1.81%

[4] H. Falk and P. Lokuciejewski, "A compiler framework for the reduction of worst-case execution times," *Real-Time Syst.*, vol. 46, no. 2, pp. 251–300, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1007/s11241-010-9101-x

[5] P. Lokuciejewski, T. Kelter, and P. Marwedel, "Superblock-based source code optimizations for wcet reduction," in *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*, ser. CIT '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1918–1925. [Online]. Available: http://dx.doi.org/10.1109/CIT.2010.327

[6] A. Yıldız, H. F. Ugurdag, B. Aktemur, D. İskender, and S. Gören, "CPU design simplified," in *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, Sep. 2018, pp. 630–632.

[7] F. Nielson, H. R. Nielson, and C. Hankin, *Principles of program analysis*. Springer, 2015.

[8] "Clang LibTooling," https://clang.llvm.org/docs/LibTooling.html.

[9] F. Stappert and P. Altenbernd, "Complete worst-case execution time analysis of straight-line hard real-time programs," *Journal of Systems Architecture*, vol. 46, no. 4, pp. 339–355, 2000.