Engineering Journal issn: 0125-8281

Article

Improved Development Cycle for 8-bit FPGA-Based Soft-Macros Targeting Complex Algorithms

Ehsan Ali[a] **and Wanchalerm Pora**[b,*]

Department of Electrical Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok, 10330, Thailand
E-mail: [a]ehssan.aali@email.com, [b,*]wanchalerm.p@chula.ac.th (Corresponding author)

Abstract. Developing complex algorithms on 8-bit processors without proper development tools is challenging. This paper integrates a series of novel techniques to improve the development cycle for 8-bit soft-macros such as Xilinx PicoBlaze. The improvements proposed in this paper reduce development time significantly by eliminating the required resynthesis of the whole design upon HDL source code changes. Additionally, a technique is proposed to increase the maximum supported data memory size for PicoBlaze which facilitates development of complex algorithms. Also, a general verification technique is proposed based on a series of testbenches that perform *code verification* using *comparison method*. The proposed testbench scenario integrates "Inter-Processor Communication (IPC), shared memory, and interrupt" concepts that lays out a guideline for FPGA developers to verify their own designs using the proposed method. The proposed development cycle relies on a chip that has Programmable Logic (PL) fabric (to hold the soft processor) alongside of a hardened processor (to be used as algorithm verifier), therefore, a Xilinx Zynq Ultrascale+ MPSoC is chosen which has a hardened ARM processor. The development cycle proposed in this paper targets the PicoBlaze, but it can be easily ported to other FPGA macros such as Lattice Mico8, or any non-Xilinx FPGA macros.

Keywords: FPGA, PicoBlaze, 8-bit soft microprocessor, software development methods, verification techniques.

ENGINEERING JOURNAL Volume 25 Issue 12
Received 6 March 2021
Accepted 12 December 2021
Published 30 December 2021
Online at https://engj.org/
DOI:10.4186/ej.2021.25.12.21

DOI:10.4186/ej.2021.25.12.21

1. Introduction

Since the introduction of Intel 8008, the first byte-oriented 8-bit microprocessor in November 1971[1] up to present time, the 8-bit hardware architecture continues to drive the computer industry in parallel to the upgraded 16/32/64/128-bit cores. In embedded systems, 8-bit Low-Pin-Count (LPC) microcontroller units (MCUs) which integrate few precision analog peripherals, configurable general-purpose I/O (GPIO), serial interface, and fast-data-bus architectures are excellent choice to capture analog signals, convert, and then condition them for signal-processing. Current MCUs run on clock speeds up to tens of megahertz which provides adequate computational power to handle modest signal processing tasks or drive complex real-time state machines [2]. Despite the competition from low-cost, low power 32-bit MCUs, 8-bit MCUs have their own target applications, and have an edge over their 32-bit counterparts when it comes to power consumption, cost, and electromagnetic interference (EMI) [3].

We can mention numerous applications for 8-bit microcontrollers, from implementing simple RGB LEDs [4] , control applications [5] [6] [7], battery-powered data acquisition [8], Maximum Power Point Tracking(MPPT)[9], to efficient cryptography [10] [11], and even implementing TCP/IP stack [12].

Another revolutionary technology in the world of embedded systems is Field Programmable Gate Array (FPGA). It is a silicon-based integrated circuit which contains an arrays of "programmable logic blocks", and a hierarchy of "reconfigurable interconnect" that allows the blocks to be connected. FPGA advantage over Application Specific Integrated Circuit (ASIC) technologies with standard cells is in their flexibility to be reprogrammed within few seconds. This allows designers to correct mistakes and perform many design iterations without undergoing through costly and lengthy fabrication process [13]. The drawback of this flexibility is that FPGA uses approximately 20 to 35 times more area than a standard cell ASIC, has a speed roughly 3 to 4 times slower, and consumes roughly 10 times as much dynamic power [14].

Initially FPGAs were used as "glue logic" [15], but later the addition of fixed-function components such as sophisticated clocking circuitry, Phase-Locked Loops (PLLs), analog-to-digital and digital-to-analog converters (ADCs and DACs) [16], hard-core processors, PCIe, SATA3, DisplayPort, Gigabit Ethernet, SD/SDIO, USB3, CAN, SPI, I2C, UART [17], and DDR memory controllers on a single chip, gave rise to "System on a Chip (SoC) FPGA" devices or "Platform FPGA" [18]. The SoC devices have opened the door for unlimited applications in all areas of digital circuit design. Hence, the implementation of an 8-bit design can be done via two mediums: 1) MCU or 2) FPGA (We exclude the other existing approach: ASIC due to its high Non-Recurring Engineering (NRE) cost [19]). If *flexibility* in design has highest priority and consequently the FPGA approach is chosen, then the next decision would be about the type of processors inside the FPGA.

FPGA embedded processor types are categorized into three groups [20]:

1. Soft-cores: Written in Hardware Description Language (HDL) without extensive optimization for the target architecture.
2. Firm-cores: Written in HDL implementations but have been optimized for a target FPGA architecture.
3. Hard-cores: Hard cores are fixed-function gate level intellectual property (IP) within the FPGA fabric.

Although the hard-core processors implemented in SoC chips run at higher clock rates, and consume less dynamic power [21], but their fixed design makes them totally inflexible for accommodating custom designs. In contrast soft-cores are easy to modify, and highly portable [20] which explains the rationality behind picking an FPGA soft-core macro as target processor in the work presented in this paper.

Currently we have several 8-bit macros available:

- Xilinx PicoBlaze [22]
- Lattice Mico8 [23]
- Navré [24] and pAVR [25]: Atmel AVR compatible 8-bit RISC hosted on OpenCores.org
- MCL86, MCL51, and MCL65 [26]: Intel 8088/8086, 8051, and MOS 6502 compatible

There are also academic-level cores such as: 8-bit interface task-oriented processor [27], 8-bit RISC core [28], 8-bit MiniMIPS [29] used for educational purposes, and soft-core with dual accumulator [30]". Among all cores mentioned, only Xilinx and Lattice are industry-level 8-bit cores, because they have adequate documentation and software development tools. Moreover, they have enough users who can find and file potential bugs. Any soft-core available on open-source websites such as GitHub.com or OpenCores.org must be treated with cautiousness. For example, we tested PauloBlaze [31], which is a plain VHDL implementation of PicoBlaze, hosted on GitHub website, and we observed that under specific circumstances it produces wrong result. We choose Xilinx PicoBlaze due to availability of Xilinx FPGA platforms in our laboratory, and we hereby refrain from comparing Xilinx with Lattice.

Maximum clock frequency of PicoBlaze reaches 105 megahertz (MHz) in Spartan-6 (-2 speed grade), and up to 238MHz can be achieved in Kintex-7 (-3 speed grade) devices [32]. High clock frequency of PicoBlaze, and the possibility of adding hardware accelerators next to the soft macros make PicoBlaze-based designs very attractive. As of the time of writing this paper the fastest 8-bit MCU runs at 100MHz (Silicon Labs MCU devices [33]) and do not include a programmable fabric for implementing custom hardware.

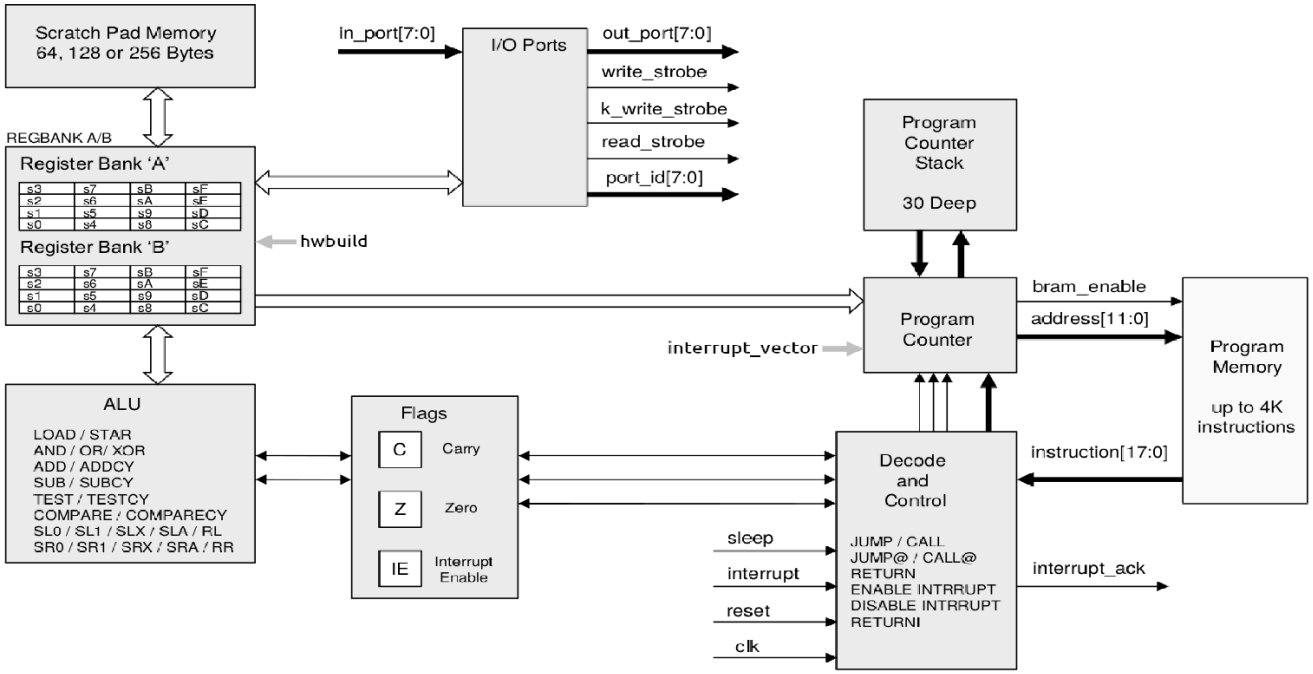

Fig. 1. KCPSM6 Architecture and Features [32].

The original standard development cycle offered by Xilinx for PicoBlaze is not suitable for developing complex algorithms which demands facilities such as step by step execution, text-based debugging output, breakpoints, etc. Although a tool (JTAGLoader [32]) provided by Xilinx can prevent resynthesizing designs when its program is modified, but it consumes a BSCAN primitive, and does not support multiple PicoBlaze cores. New FPGA boards, and new version of Vivado are not supported either, as it relies on obsolete ISE libraries. Additionally, the development cycle only opted for Xilinx devices, and does not work with other FPGA vendors such as Intel (former Altera), Microsemi (former Actel), and Lattice.

The **contributions** of this paper are: 1) **Resynthesis elimination** of FPGA designs when source code of a soft-core processor changes. This reduces developmental time significantly. 2) Resynthesis elimination **support for multi-core soft-core** architectures. This allows multiple instances of e.g., PicoBlaze to be used in a design and their programs to be modified on run-time. 3) Resynthesis elimination **does not consume any BSCAN primitive**. This frees up a BSCAN primitive and allows resynthesis elimination to be implemented on high-end FPGAs which their BSCAN primitives are already consumed by on-board components. 4) The proposed resynthesis elimination mechanism **works for both Xilinx and non-Xilinx FPGA platforms**. 5) A **verification mechanism** to ensure the integrity of complex algorithms written for FPGA-based soft-macros.

Note that our proposed method can be used in any soft-core that relies on internal FPGA memory blocks. For example, one can take advantage of our proposed method and avoid FPGA re-synthesis when a low-end FPGA design uses the 32-bit Xilinx MicroBlaze soft-core and replaces external DDR3 or flash memories with internal BRAMs.

This paper is organized into nine sections. After this introduction, the second section explains the PicoBlaze, its application, and its standard development cycle provided, and its flaws are pointed out. In third section the assembler for PicoBlaze is discussed. Section Four explores available simulation options such as FIDEx [34]. In Section Five, necessary software, and hardware combination needed to improve the development cycle is proposed. In Section Six an address generator circuitry is introduced to provide designers with a shared data memory between the ARM core, and the PicoBlaze. In Section Seven, code verification using *comparison method* is employed to perform the "equivalence checking" of the developed PicoBlaze program versus the result obtained from C language program running on ARM core. Section Eight provides the analysis of conserved development time using the proposed method. Finally, we conclude our article in the last section.

2. The PicoBlaze (KCPSM6) Macro

2.1. Overview

The latest version of PicoBlaze is technically called KCPSM6 which is derived from older version "(K)constant Coded Programmable State Machine 3" (KCPSM3) [35]. It is a soft macro which defines an 8-bit data processor that can execute a program of up to 4K instructions. All instructions have 18-bit width and all of them need 2 clock cycles for execution. It provides two banks of 16 general purpose registers [32]. The KCPSM6

DOI:10.4186/ej.2021.25.12.21

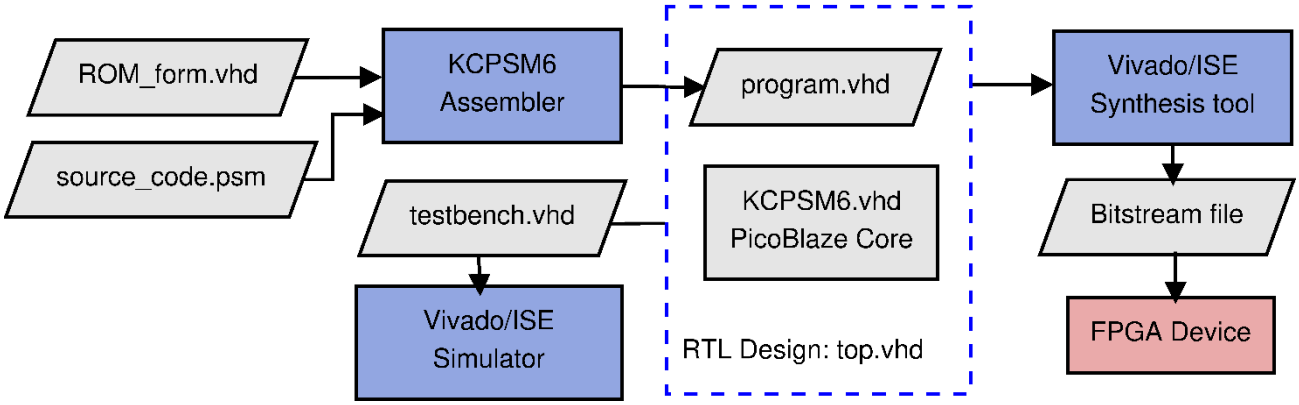

Fig. 2. PicoBlaze Standard Development Cycle.

architecture overview as provided in official user manual is shown in Fig. 1.

The soft-core provides the following facilities:

1. 6-bit opcode field that enables the support of up to 64 instructions. The original KCPSM6 Instruction Set Architecture (ISA) has 55 instructions.
2. Thirty-two 8-bit registers which are organized in two banks 'A' and 'B'. Only one bank can be activated at any given time.
3. Maximum 256-byte Scratch Pad Memory (SPM) that works as data memory.
4. A stack with depth of 30.
5. 12-bit address bus which covers 4KB of program memory.
6. 256 input ports, and 256 output ports.
7. One interrupt signal.

The PicoBlaze cannot support high-level programming languages effectively due to its simplicity [36]. Therefore, most designs use the assembly language for implementing algorithms.

2.2. PicoBlaze Applications

We can mention myriad examples of PicoBlaze Applications. We have PicoBlaze used in embedded systems for "monitoring applications" [37], Vladimir employed the processor to provide a controller for traffic light [38], Pavel constructed a multiprocessor parallel architecture based on message passing paradigm using multiple PicoBlaze cores [39], Venkata studied the usage of the PicoBlaze in "multiprocessor systems" [40], and Robert implemented a network interface using the PicoBlaze [41]. Lung, Sabou, and Barz implemented "smart sensor using multiple cores" of PicoBlaze [42].

Seema and Purushottam used PicoBlaze to implement a "wireless sensor network" [43]. PicoBlaze is used as a "configuration engine" in a fault-tolerance technique [44]. Hassan and Benaissa implemented a scalable elliptic curve cryptography (ECC) on PicoBlaze [45]. Tim Good and Benaissa used PicoBlaze for "advanced encryption standard" (AES) [46]. This body of literature justifies the usage of 8-bit soft-core processors such as PicoBlaze in a broad range of applications.

2.3. Standard Development Cycle - Related Work

The only related work to this paper is the standard development cycle provided by Xilinx, which will be discussed in detail in this section. The standard development cycle for the latest version of PicoBlaze (KCPSM6) provided by Xilinx is shown in Fig. 2. The steps for VHDL language are listed below (Verilog language is also supported) [32]:

1. Import "KCPSM6.vhd" (PicoBlaze core VHDL version) into ISE [47] or Vivado [48] project.
2. Write a PicoBlaze program and save the source code into a "source_code.psm" file.
3. Select an appropriate "ROM_form.vhd" that matches target FPGA Xilinx device.
4. Run assembler on "source_code.psm" and "ROM_form.vhd" files and get "program.vhd" as assembler output.
5. Import "program.vhd" into ISE or Vivado project.
6. Connect both KCPSM6 and program modules together inside a wrapper module ("top.vhd") using signals.
7. Run ISE or Vivado simulator and debug the program by looking into registers and SPM content by examining simulation signals and waveforms.
8. Synthesize the complete design, and upload generated bit-stream file into FPGA device.

2.4. Standard Development Cycle Limitations

The Xilinx PicoBlaze standard tools fall short when it comes to complex programming tasks. The only way to check the register content and SPM memory is through Vivado/ISE simulator waveform which is not practical if the program is more than few hundred lines. Adding more instructions between lines or simply a change in conditional jumps, modifies the simulation timing and makes waveform-based debugging very challenging. Other issues are lack of breakpoints, and step by step

Table 1. PicoBlaze Assemblers.

Assembler	Supported Cores	Host OS	Status	License	Features
Xilinx [52]	KCPSM3, KCPSM6	Windows Linux(wine)	v2.7 Stable	Xilinx	Outputs .fmt, .log, .hex Open PicoBlaze [51]
Open PicoBlaze [51]	KCPSM3, KCPSM6	Any OS with Python	v1.3 Stable	Free MIT license	High performance, m4 preprocessor, static code analysis, local labels

Table 2. PicoBlaze Simulators.

Simulator	Supported Cores	Host OS	Status	License	Features
kpicosim [53]	KCPSM3	Linux	v0.7 Beta	Free	Syntax highlighting, compiler, simulator, and export functions to VHDL, HEX and MEM files.
sc0ty [54]	KCPSM3	Linux	Beta	GNU GPL	wxWidgets library based, supports LED, switches, keyboard, terminal, and timer.
FIDEx [34]	KCPSM3, KCPSM6, Mico8	Linux, Windows	2016-09.0 Stable	Proprietary	Project manager, memory page support, full-fledged debug facility

execution. Meanwhile, the mandatory resynthesis step and the need to reupload the bitstream file into FPGA device increase the development time significantly.

In normal design flow, designer imports "program.vhd" to a Vivado/ISE Design Suite project, and then synthesizes the design, and finally uploads the generated bitstream into the FPGA board. The problem with this approach is that whenever PicoBlaze program is modified, a rerun of assembler to generate a new "program.vhd" is required. The change in content of "program.vhd" file triggers a complete resynthesis of wrapper module that holds the "program" Block RAM module. To solve this, the "JTAG Loader" [32] is provided by the Xilinx. It is a tool designed to upload the generated .hex file by assembler to program BRAM. It eliminates the need to resynthesize the design. Some shortcomings of the tool are mentioned below:

- Only one PicoBlaze core (marked with "C_JTAG_LOADER_ENABLE => 1" generic) in the design is supported.
- Depends on old drivers provided by ChipScope [49] and needs ISE Design Suite to be installed.
- No support for new advanced development boards such as "Xilinx ZCU104" [50] that has several devices attached to its JTAG chain.
- Consumes a BSCAN primitive.

Another issue is lack of support for other FPGA vendors. PicoBlaze core and all its development tools target Xilinx devices only and cannot be ported easily to other FPGA devices manufactured by other vendors such as Intel, Microchip, and QuickLogic.

The rest of paper covers several proposed techniques needed to solve all issues mentioned above by integrating third party tools with standard Xilinx tools to form a reliable and consolidated solution for implementing complex algorithms on PicoBlaze.

3. PicoBlaze Assembler

Currently, there are only two reliable PicoBlaze assemblers which are listed in Table 1. The original Xilinx assembler receives a program source code with extension .psm and outputs a formatted PSM File .fmt, a .log file, a .hex file which contains raw equivalent hex value of each instruction, and a .vhd file if "ROM_form.vhd" template file exists. In most cases the original assembler is sufficient. Open PicoBlaze Assembler (Opbasm [51]) is an alternative option which offers special features such as faster assembling time, m4 preprocessor macros, static code analysis to identify dead code and optionally remove it, code block annotations with user defined PRAGMA meta-comments, and the support for local labels. In this paper, the original KCPSM6 assembler is chosen as it exhibits acceptable degree of stability and is used widely by the community.

4. PicoBlaze Simulator

The standard waveform-based simulator suffices for simple algorithms that can be implemented with less than one or two hundred instructions. Anything more complex needs a full-fledged simulator with breakpoints, step by step execution, registers, and SPM content monitoring capabilities.

An exhaustive search for all available PicoBlaze simulators yields few results. Those which were buggy, unstable, or had no proper documentation were omitted. Table 2 shows those simulators which have passed the

following criteria: A stable version is available, a Graphical User Interface (GUI) is provided, debugging facilities such as step by step execution and breakpoints are available, proper documentation for compiling the source and using the tool itself is provided. We found the FIDEx the only solid simulator which supports the latest version of PicoBlaze (KCPSM6). All other simulators are either out of date or only support KCPSM3, or lack quality, or a crucial debugging functionality.

5. Improved Development Cycle for PicoBlaze

For implementing a complex algorithm on PicoBlaze the suggested development method which is: "To debug using functional simulation or running the program on hardware directly [32]" will not suffice. Our proposed development setup includes an isolated PicoBlaze core (pBlaze) on Program Logic (PL) of an FPGA connected to standard URAT modules. The pBlaze module consist of three submodules: 1) kcpsm6.vhd (the core provided by Xilinx) 2) tx6.vhd (UART Send) 3) rx6.vhd (UART Receive). The program block memory (pBlaze prog.vhd) is moved to outside of the pBlaze VHDL module, The dual port mode on the program block RAM (BRAM) is enabled with the following IP settings: Mode= "Stand Alone Memory", Type= "True Dual Port RAM", Primitive Type= "BRAM".

The read/write width, and memory depth setting is shown in Fig. 4. Development starts in any IDE which provides a simulator (such as FIDEx IDE [34]) by writing assembly code. FIDEx supports several other processors beside PicoBlaze (e.g., Lattice Mico8) and has its own assembler dialect. The FIDEx dialect is used to implement an algorithm, and its simulator is invoked to verify algorithms correctness. Next, we convert the developed machine code in FIDEx assembly dialect to original KCPSM6 syntax using a *sed* [55] script shown in Listing 1. The script outputs a new .psm file (PicoBlaze assembly source code) which then can be fed into standard KCPSM6 assembler.

5.1. Proposed Hardware Architecture

Any Xilinx SoC FPGA which incorporates a processor next to an FPGA fabric can be chosen as development platform. Note that our proposed method is around controlling the synthesized hardware on FPGA using a hardened processor. This hardened core does not need to be resynthesized as it is fixed, therefore, it can be used to rewrite (dynamic resynthesis) parts of the synthesized hardware on FPGA PL fabric. This method can be employed in any FPGA SoC device that is equipped with a hardened processor such as Intel Stratix10 FPGA that offers a quad core 64-bit ARM Cortex-A53 or Microchip SmartFusion ProASIC3 IGLOO that has a hard 100Mhz ARM Cortex-M3 core. Interestingly, in case that a hardened core is absent, our proposed method is still viable as designers can replace

Listing 1. Sed script to convert FIDEx dialect to KCPSM6.

```
s /\bRET\b/RETURN/ g
s /\bCOMPC\b/COMPARECY/ g
s /\bCOMP\b/COMPARE/g
s /\bTESTC\b/TESTCY/ g
s /\bADDC\b/ADDCY/ g
s /\bSUBC\b/SUBCY/ g
s /\bROLC\b/SLA/ g
s /\bRORC\b/SRA/ g
s /\bLOADRET\b/LOAD&RETURN/ g
s /\bRDMEM\b/FETCH/ g
s /\bRDPRT\b/INPUT/ g
s /\bWRPRT\b/OUTPUT/g
s /\bWRMEM\b/STORE/ g
s /0 x // g
```

the hardened processor with a soft-core. For example, Microchip PolarFire offers a Soft RISC-V or Soft ARM Cortex-M1 as processing option which can be used as Processing System (PS).

The "Xilinx Zynq Ultrascale+" device is chosen as it provides a PS alongside of a Programmable Logic (PL). The Vivado Design Suit 2018.3 [48] is used to create a project that demonstrates the proposed improved development cycle for PicoBlaze. The Vivado project consist of a main "Vivado Block Design" (BD) named "system.bd". The system BD schematic is shown in Fig. 3. At the heart of the BD resides a ZYNQ UltraScale+ MPSoC which manages data transfer between all these components via AXI interconnects: two shared Block RAMs, a PicoBlaze core, and hardened ARM processor.

Figure 3 shows simplified schematic of components inside the BD. Both Zynq Ultrascale+ MPSoC and PicoBlaze are equipped with UART send and receive ports which boost the debugging process by providing terminal input and output for both processors. Registers value, memory locations, and program variable can be dumped to terminal through designated serial ports. One of the two block RAMs contains the PicoBlaze program and the other one acts as a shared data memory. Next section discusses required BRAM setting for the proposed setup. Full Vivado project is available at author's GitHub public domain [56].

5.2. Memory Block RAMs

Two Block RAMs (BRAMs) are used in proposed development cycle. One holds PicoBlaze program while the other one shares data between PicoBlaze and ARM cores. This shared channel is used for verifying algorithms implemented on PicoBlaze with the ARM processor as verifier unit.

5.2.1. PicoBlaze Program BRAM

The following calculation must be considered to set a dual port PicoBlaze program BRAM memory

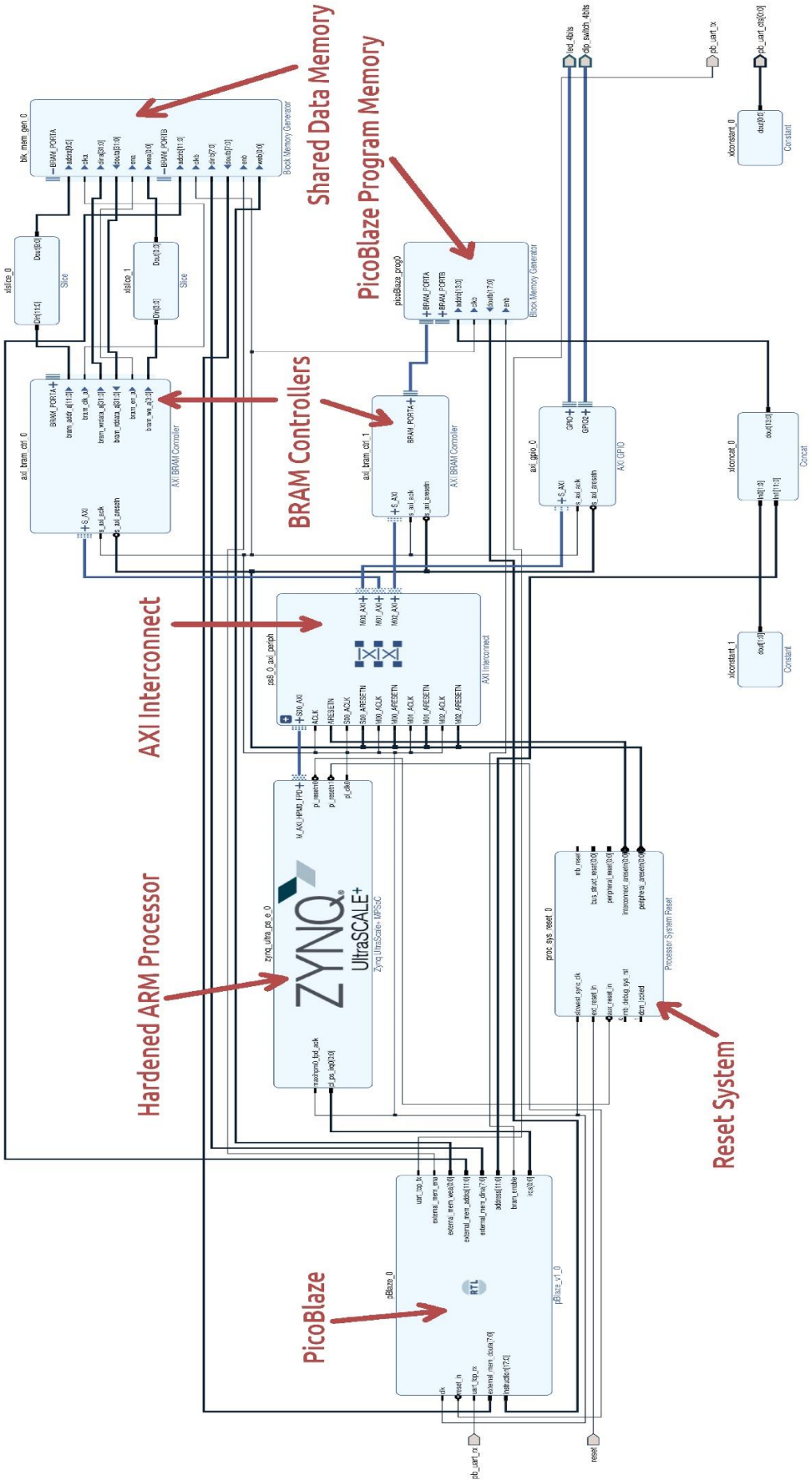

Fig. 3. Vivado Block Design of PicoBlaze Development Environment.

DOI:10.4186/ej.2021.25.12.21

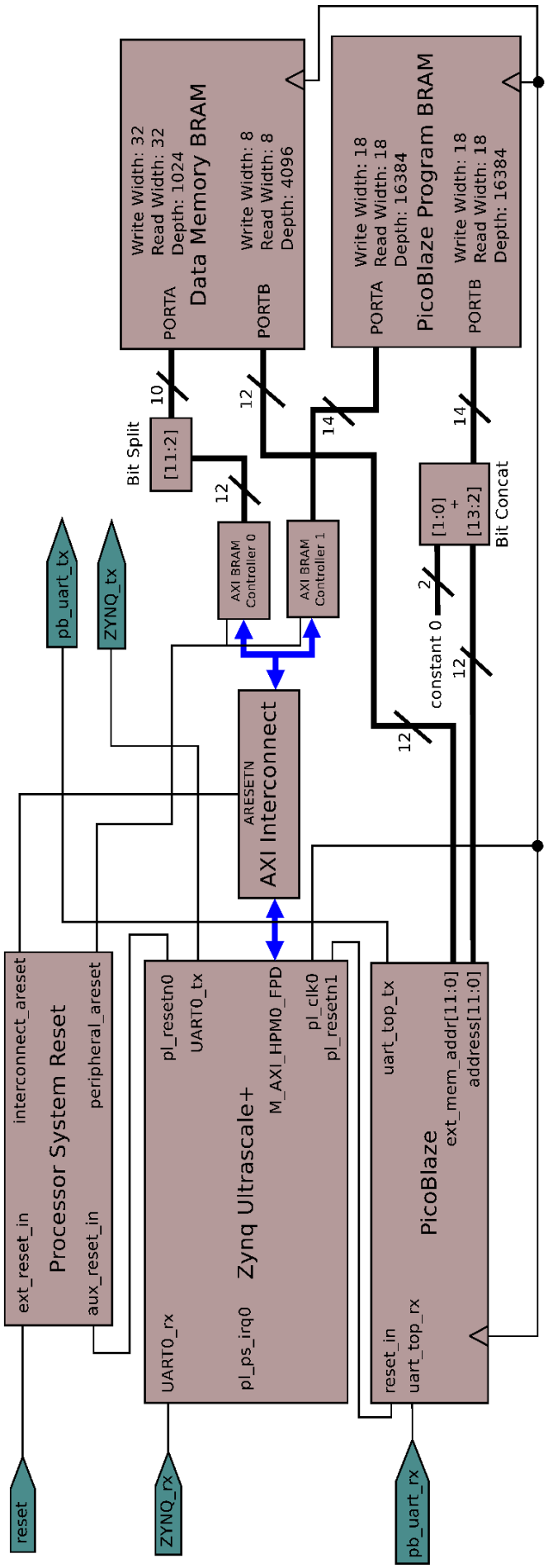

Fig. 4. Zynq Ultrascale+ and PicoBlaze Hardware Platform.

Listing 2. fill_picoBlaze_BRAM() function [runs on FPGA PS]

```
void fill_picoBlaze_BRAM ( ) {
  int  loc = 0 ;
  for (int i =0; i <16384; i=i +4) {
    Xil_Out32 (
      XPAR_AXI_BRAM_CTRL
        _1_S_AXI_BASEADDR + i,
      program_4k [ loc ] ) ;
    loc++;
  }
}
```

specification: PicoBlaze has a 12-bit address bus, therefore, 2^{12} = 4096 locations can be addressed. Its instruction width is 18-bit; therefore, the memory size must be 18 ∗ 4096 =73728 bits or 72 kbit = 9kB. The PicoBlaze core is not the only module that accesses this BRAM. The ARM processor through AXI interconnect also must be able to perform read and write memory operations to and from this BRAM. The AXI interconnect supports only 32-bit data-width, therefore, demanding a 32 ∗ 4096 = 131072 bits or 128 kbit = 16kB BRAM.

The conclusion is that although program BRAM needs only 9KB, but AXI interconnect forces us to assign 16KB resulting in 16 − 9 = 7kB memory will be wasted. Figure 4 shows the width of PORTA and PORTB of the program BRAM is 14-bit. This provides the ability to address 16384 locations. A 2-bit logical left shift of address bus is required for 4-byte alignment of PicoBlaze 12-bit addresses. Note that write and read width of both ports are 18- bit.

5.2.2. Data Memory BRAM

A dual port BRAM is used to share information between two systems. The size of RAM is 4098 ∗ 8 bits. Port A is 10-bit wide and connected to the ARM processor via AXI interconnect. This gives access to 1024 memory location. ARM processor can access the whole 4KB memory by reading or writing 32-bit per memory access. The port B is 12-bit wide and is connected to PicoBlaze with 8-bit read and write width.

5.3. Proposed Software Architecture

5.3.1. ARM Application Project

After synthesizing the design proposed in previous section in Vivado Design Suite, we export the *hardware platform* to Xilinx SDK. We create a C language *Application Project* for target processor psu_cortexa53_0

Listing 3. hex2ch.cpp Tool [runs on development environment].

```
// This program converts picoblaze's .hex to //
SDK .h header file.
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main (int argc, char *argv[]) {
   if (argc < 2) return -1;
   string input_filename = argv[1];
   // --------------------------------
   // Extract the filename by removing
   // the extension
   size_t lastindex = input_filename.
   find_last_of(".");
   if (lastindex == string::npos)
      return -2;
   string rawname = input_filename.substr
                        (0, lastindex);
   // --------------------------------
   // Add .h extension to input filename
   string output_filename = rawname + ".h";
   ifstream file_in (
   input_filename.c_str(), ios::in);
   ofstream file_out (
   output_filename.c_str(), ios::out);
   file_out << "u32 program_4k[4096]={" <<
      endl;
   string l;
   string line;
   // --------------------------------
   // Get the first line
   getline(file_in, l);
   if (l.size() && l[l.size()-1]=='\r')
      line = l.substr( 0, l.size() - 1 );
   else
      line = l;
   file_out << "0x" << line << endl;
   // --------------------------------
   // iterate through the remaining
   // lines.
   while (getline(file_in, l)) {
      if (l.size() && l[l.size()-1]=='\r')
         line = l.substr( 0, l.size() - 1 );
      else
         line = l;
      file_out << "," << "0x" <<
      line << endl;
 }
 file_out << "};" << endl;
 return 0;
}
```

with OS Platform option set to *standalone*. The project name is "picoblaze_app" and its source code can be found under "picoblaze_dev.sdk" folder [56]. The entry point is "main.c" which included the header file "pBlaze_prog.h". The "pBlaze_prog.h" file defines a 4K C language array that contains hex value of instructions designed to be uploaded to PicoBlaze program BRAM memory.

The utility function fill_picoBlaze_BRAM() which is defined in "main.c" is used to upload a PicoBlaze program into the BRAM memory controlled by AXI_BRAM _CTRL_1. It performs the task by reading a one-dimensional u32 array with the size 4096 of bytes (program_4k) and writes it into program BRAM. The function source code is shown in Listing 2.

5.3.2. Hex to Header File Utility

The *program_4k* array is defined in "pBlaze_prog.h" header file and must be regenerated every time the designer modifies the PicoBlaze's program. This header file must be included in the "main.c". Listing 3 shows the C++ source code for "hex2ch.cpp" file. It is a command line utility written by authors to perform the conversion between PicoBlaze hex file generated by KCPSM6 assembler to "pBlaze_prog.h" header file.

To compile we issue the command "$ g++ -o hex2ch hex2ch.cpp", and to convert pBlaze_prog.hex we issue: "$ ./hex2ch pBlaze_prog.hex" which outputs "pBlaze_prog.h" header file in current working directory.

5.4. Proposed Development Cycle

With discussed hardware platform and software tools, a complete development cycle for PicoBlaze that can handle complex algorithms can now be achieved. To develop for PicoBlaze we propose the following steps:

1. Synthesize the hardware platform and export it to Xilinx SDK.
2. Program the FPGA using the synthesized hardware.
3. Edit source code in FIDEx ("program.psm" file).
4. Simulate the code in FIDEx.
5. Run sed script on "program.psm" file. (Output is "program_pb.psm" file)
6. Run KCPSM6 assembler on program_pb.psm.(Output is "program_pb.hex")
7. Run hex2ch on program_pb.hex. (Output is "program_pb.h")
8. Update "program_pb.h" that resides in SDK folder.
9. Run SDK Application on FPGA to update the PicoBlaze program in FPGA.

Any modification to PicoBlaze program (Step #3) triggers the rerun of steps #5 to #8 which can be easily scripted in user development machine (e.g., a Linux bash

DOI:10.4186/ej.2021.25.12.21

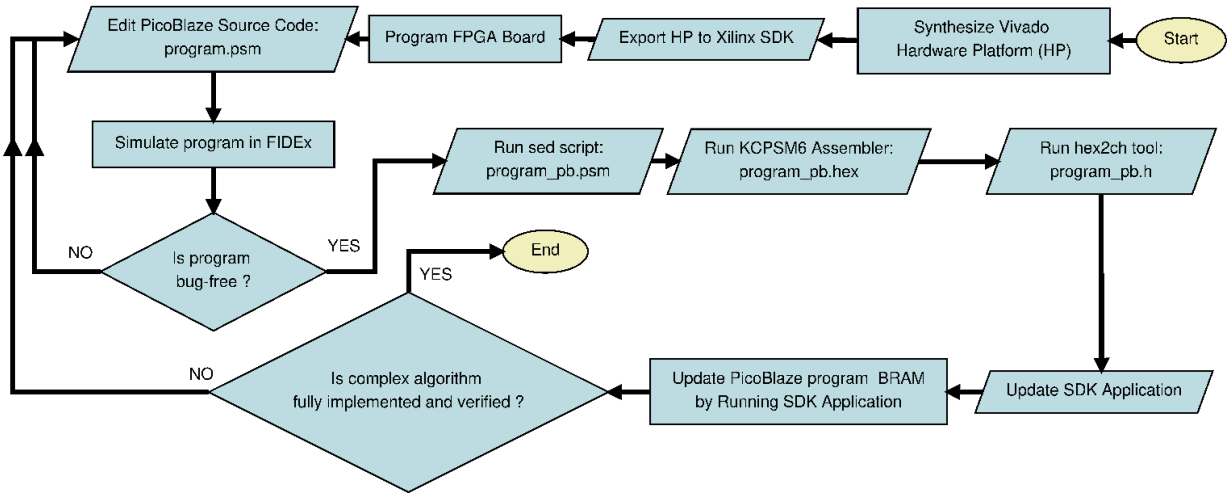

Fig. 5. Improved Development Cycle for PicoBlaze Macro.

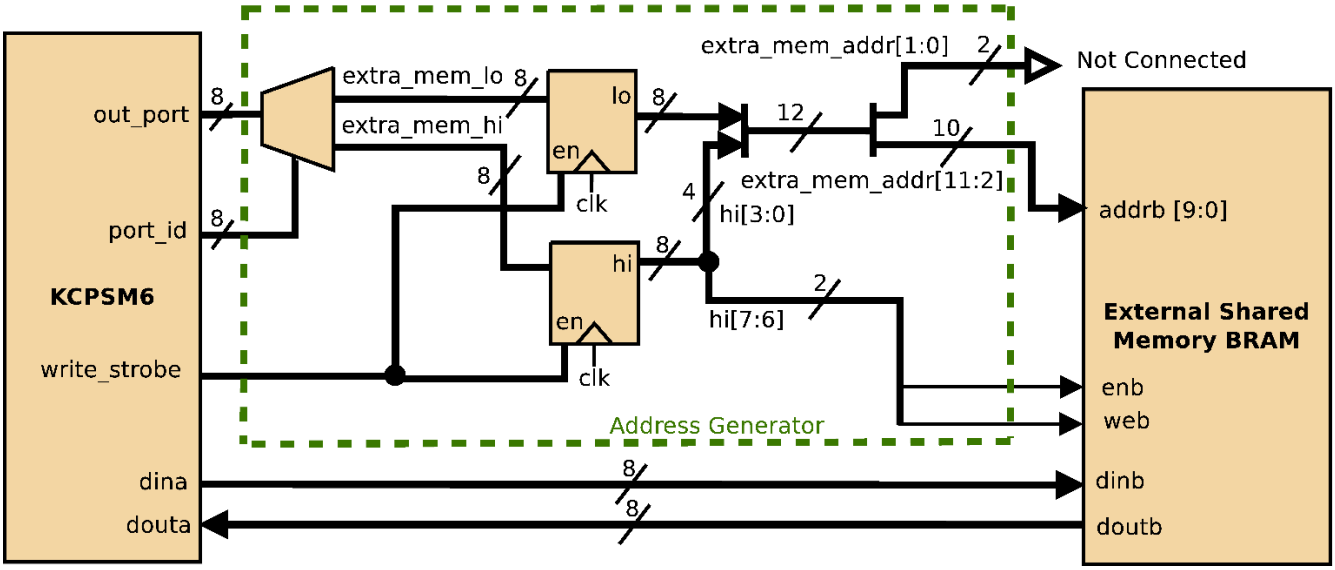

Fig. 6. PicoBlaze Address Generator Circuit used for Data Memory Expansion,

script). Figure 5 shows the complete flowchart of improved development cycle for PicoBlaze macro.

6. Proposed Address Generator Circuitry

The PicoBlaze's 12-bit address supports maximum range of 4KB program memory. Its SPM which is used as data memory can have maximum size of 256 bytes.

To add another 4KB BRAM as a shared data memory to PicoBlaze-based systems, an address generator circuit is designed and shown in Fig. 6. The design requires 7 instructions, or 14 clock cycles to read/write a byte from/to shared data memory locations. Accessing this BRAM is 7 times slower than the main program BRAM. To access the memory, two routines are provided: Read_ext_mem() and Write_ext_mem() which are defined in Listing 4. The programmer simply calls these two routines whenever a memory access to the 4KB data memory is required.

The s6, and s5 are general purpose PicoBlaze registers. For reading from memory, the register pair [s6, s5] is used with s6 as high byte, and s5 as low byte. The 12-bit read address is shown by 'A' letters. The bit 7 of s6 is Clock Enable (represented by letter 'C') and register s7 holds the read data. The 16-bit register pair format is shown below:

8-bit s7 register	8-bit s6 register	8-bit s5 register
READ DATA	C000_AAAA	AAAA_AAAA

Similarly, for writing to memory, a 12-bit address is formed in [s6, s5] register pair. The bit 7 of s6 is Clock Enable, and bit 6 of s6 is Write Enable, and are represented by letters 'C', and 'W'. The register s7 must contain 8-bit write data. The 16-bit register pair format is shown below:

8-bit s7 register	8-bit s6 register	8-bit s5 register
WRITE DATA	CW00_AAAA	AAAA_AAAA

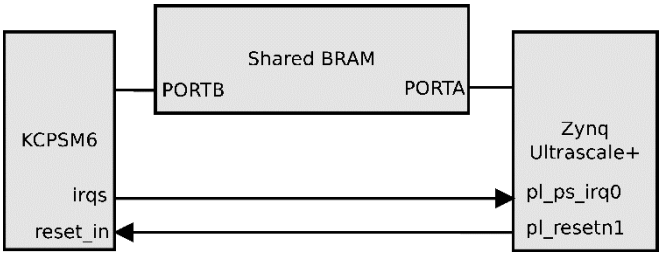

Fig. 7. PicoBlaze & Zynq Ultrascale+ Inter-Processor Communication Platform for Verification.

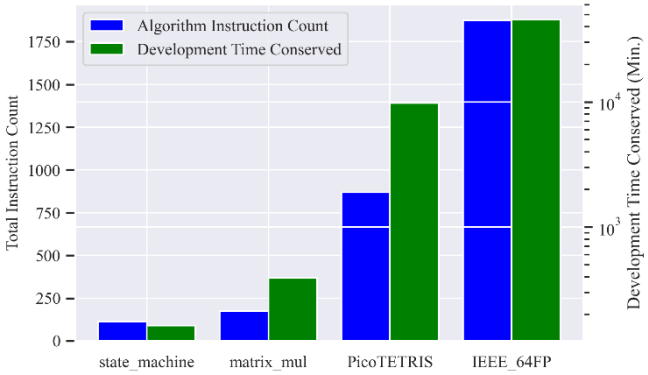

Fig. 8. Four Verification Algorithms and Development Time Conserved using our proposed method

7. Proposed Verification Mechanism

7.1. Concepts

Verification is the process of determining that a model implementation accurately represents the developer's conceptual description of the model and the solution to the model [56] [57]. Verification can be classified into: A) *Code Verification*: To identify and eliminate programming and implementation errors within the software B) *Calculation Verification*: to quantify the error of a numerical simulation or in other words "numerical error estimation" [57]. A widely used approach in code verification is the *comparison method* in which one code is compared to another established code [58].

In our proposed method the already established code resides in the PS side of FPGA. It can be an already established code (e.g., a well-known C language library for ARM Cortex A-53 of Zynq Platform), or a hardware module that is available in the PS side such as the VFPv4 hard unit inside ARM processor. This unit is fully IEEE-754 compliant [59]. An Inter-Processor Communication (IPC) [60] is established based on shared-memory, and interrupt signaling as shown in Fig. 7.

Take implementing a 64-bit floating-point library (complex algorithm) on PicoBlaze as an example [61]. After writing routines that perform floating point operations in the assembly language of PicoBlaze, we can verify the result by first writing the PicoBlaze floating point arithmetic result into the "Shared BRAM" and then compare the result with of those that are produced by ARM processor (either by a software library or hardware floating point unit).

Listing 4. Shared Memory Read/Write Routines

```
CONSTANT Extra_mem_hi_output_port, 02
CONSTANT Extra_mem_output_port,    03

Read_ext_mem:
  OR     s6, 80  ;Enable BRAM clock
  OUTPUT s5, Extra_mem_lo_output_port
  OUTPUT s6, Extra_mem_hi_output_port
  OR     s5, s5  ;Delay
  INPUT  s7, Extra_mem_input_port
  AND    s6, 7F  ;Disable BRAM clock
  OUTPUT s6, Extra_mem_hi_output_port
RETURN

Write_ext_mem:
  ;Enable BRAM and write enable.
  OR     s6, C0
  OUTPUT s7, Extra_mem_output_port
  OUTPUT s5, Extra_mem_lo_output_port
  OUTPUT s6, Extra_mem_hi_output_port
  OR     s5, s5     ;Delay
  ;Disable BRAM and write enable.
  AND    s6, 3F
  OUTPUT s6, Extra_mem_hi_output_port
RETURN
```

To provide more verification scenarios beside IEEE_64FP, three additional programs are chosen: 1) A real-time complex state machines controller that manage multi-core systems like the work presented in [62]. 2) A loop in loop matrix multiplication algorithm. 3) A Tetris game written for PicoBlaze [63].

Figure 8 shows four selected algorithms for the verification process. The blue bars are total instruction count per algorithm and the green bars indicate the time conserved in development time in minutes. Notice the positive correlation between the number of machine instructions and conserved time. Note that the y-axis for the conserved time is set to logarithmic scale to overcome the visualization of data with wide range gap.

The conserved time is calculated based on average number of resynthesis required during the development of each algorithm. For state_machine 163 minutes, matrix_multiplication 393 minutes, PicoTETRIS 9793 minutes and IEEE_64FP 45540 minutes are conserved.

The equivalent version of each algorithm is written in C and then the verification method proposed in this paper is employed to ensure the correctness of the implementation. All results point to reliability of our proposed method.

7.2. Mechanism

In this section the details of IEEE64_FP algorithm verification is provided. Initially, the ARM core writes

Table 3. Synthesis and Implementation Elapsed Time.

Run	1st	2nd	3rd
Launch Runs	01:07	00:29	00:33
Submodule Runs	08:56	00:00	00:00
System Synthesis	01:14	02:03	01:42
System Implementation	04:35	05:54	05:34
Total	**15:52**	**08:26**	**07:49**

* All time periods are in MM:SS format where M stands for Minute and S stands for Second.

input data into the shared memory and resets the PicoBlaze core (by asserting pl_resetn1 signal in Fig. 7). PicoBlaze reads the input data written by ARM through routines defined in previous section, and performs the operations specified in algorithm (e.g. addition), and then writes the result back to the shared BRAM memory.

Next, PicoBlaze calls invoke_done_interrupt() routine to send an interrupt to ARM core (by asserting pl_ps_irq0 signal in Fig. 7) to indicate the end of calculation. The ARM core then reads the calculated result and compares it with of its own generated results.

The verification loop then replaces the input data and rewrites it into shared memory and resets the PicoBlaze again. It keeps comparing the result until all testcases pass. Finally, a list of all failed cases is printed, or else it outputs a "verification passed" message to ZCU104 serial debugging output port.

8. Synthesis and Implementation Elapsed Time Analysis

In this section the synthesis and implementation time of proposed architecture are measured, and the time conserved via the proposed method is calculated. The Xilinx Vivado v2020.1 (64-bit) is used on a development machine running Windows 10 64-bit operating system. The processor is Intel i7-8750 @ 2.30HHz with 16 GB of installed RAM. Six cores are assigned to Vivado runs and the elapsed times to synthesize the design for Xilinx Zynq Ultrascale+ MPSoC ZCU104 Evaluation Kit are shown in Table 3.

Initially, the TCL "reset_project" command is issued to clear all the outputs of previous runs and then synthesis and implementation commands are issued. In 1st run the total time needed to have the design up and running is 15 minutes and 52 seconds. After applying a modification to PicoBlaze program the current synthesis becomes out of date and a resynthesis is necessary. The Vivado suite is intelligent enough to recognize the intact submodules and excludes them from resynthesis process. That is why the time elapsed for submodule runs in 2nd run is zero.

Notice that the total required time to setup the design is reduced from 15:52 to 08:26 in 2nd run and 07:49 in 3rd run. The time reduction in 3rd run is due to system caching. From Table 3. It can be seen that designers must wait for about 8 minutes to synthesize and implement their design into development board every time a line of PicoBlaze code changes.

9. Conclusion

In this paper an improved development cycle for PicoBlaze is proposed. It integrates a simulator with assembler and eliminates the FPGA resynthesis whenever programmer changes the source code of soft-core. The proposed method supports multi-core PicoBlaze architecture and does not rely on BSCAN primitives and JTAG communication, but AXI interconnect IP core. Additionally, a verification mechanism is proposed which enables designers to verify their PicoBlaze code against already established libraries or hardware units. Another proposed improvement is the expansion of PicoBlaze SPM size through introducing a 4KB shared memory controlled by an address generator circuitry.

Acknowledgment

This research is supported financially by "The Chulalongkorn Academic Advancement into Its 2nd Century Project". The student is awarded a joint scholarship, composed of "The 100th Anniversary Chulalongkorn University Fund for Doctoral Scholarship" and "The 90th Anniversary of Chulalongkorn University, Rachadapisek Sompote Fund".

References

[1] S. P. Morse, B. W. Ravenel, S. Mazor, and W. B. Pohlman, "Intel Microprocessors–8008 to 8086," *Computer*, vol. 13, no. 10, pp. 42–60, Oct. 1980.

[2] J. Yiu, "Software based finite state machine (FSM) with general purpose processors," ARM - White Paper, Jan. 2013.

[3] "Application ideas for 8-bit low-pin-count microcontrollers," Aug 2011. [Online]. Available: https://www.fujitsu.com/downloads/MICRO/fma/formpdf/LPC-TB_071009.pdf

[4] Y. Yang, "Implementation of a colorful RGB-LED light source with an 8-bit microcontroller," in *20105th IEEE Conference on Industrial Electronics and Applications*, Jun. 2010, pp. 1951–1956.

[5] C. . Hsu, I. . Chung, C. . Lin, and C. . Hsu, "Selfregulating fuzzy control for forward DC-DC converters using an 8-bit microcontroller," *IET Power Electronics*, vol. 2, no. 1, pp. 1–12, Jan. 2009.

[6] D. He and R. M. Nelms, "Peak current-mode control for a boost converter using an 8-bit microcontroller," in *IEEE 34th Annual Conference on Power Electronics Specialist*, PESC '03, June 2003, pp. 938–943, vol. 2.

[7] H. S. Khan and M. B. Kadri, "DC motor speed control by embedded PI controller with hardware-in-loop simulation," in *2013 3rd IEEE International*

Conference on Computer, Control and Communication (IC4), Sep. 2013, pp. 1–4.

[8] R. Mukaro and X. F. Carelse, "A microcontrollerbased data acquisition system for solar radiation and environmental monitoring," *IEEE Transactions on Instrumentation and Measurement*, vol. 48, no. 6, pp. 1232–1238, Dec. 1999.

[9] S. Oprea, M. Rosu-Hamzescu, and C. Radoi, "Implementation of simple MPPT algorithms using low-cost 8-bit microcontrollers," in *Proceedings of the 2014 6th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, Oct. 2014, pp. 31–34.

[10] Z. Liu, T. Pöppelmann, T. Oder, H. Seo, S. S. Roy, T. Güneysu, J. Großschädl, H. Kim, and I. Verbauwhede, "High-performance ideal lattice-based cryptography on 8-bit AVR microcontrollers," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 4, pp. 117:1–117:24, Jul. 2017. [Online] Available: http://doi.acm.org/10.1145/3092951 (accessed Mar. 4, 2021).

[11] S. C. Seo and H. Seo, "Highly efficient implementation of NIST-compliant Koblitz curve for 8-bit AVR-based sensor nodes," *IEEE Access*, vol. 6, pp. 67 637–67 652, 2018.

[12] A. Dunkels, "Full TCP/IP for 8-bit architectures," in *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services,ser. MobiSys '03*, New York, NY, USA, ACM, 2003, pp. 85–98. [Online]. Available: http://doi.acm.org/10.1145/1066116.1066118

[13] I. Kuon, R. Tessier, and J. Rose, *FPGA Architecture: Survey and Challenges*. Now, 2008. [Online]. Available: https://ieeexplore.ieee.org/document/8187326

[14] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, Feb. 2007.

[15] B. Fawcett, "FPGAs as reconfigurable processing elements," *IEEE Circuits and Devices Magazine*, vol. 12, no. 2, pp. 8–10, Mar. 1996.

[16] A. Zanikopoulos, P. Harpe, H. Hegt, and A. Van Roermund, "A flexible ADC approach for mixed-signal SoC platforms," in *2005 IEEE International Symposium on Circuits and Systems*, May 2005, pp. 4839–4842, vol. 5.

[17] S. Ahmad, V. Boppana, I. Ganusov, V. Kathail, V. Rajagopalan, and R. Wittig, "A 16-nm multiprocessing system-on-chip field-programmable gate array platform," *IEEE Micro*, vol. 36, no. 2, pp. 48–62, Mar. 2016.

[18] S. Anvar, O. Gachelin, P. Kestener, H. Le Provost, and I. Mandjavidze, "FPGA-based system-on-chip designs for real-time applications in particle physics," *IEEE Transactions on Nuclear Science*, vol. 53, no. 3, pp. 682–687, Jun. 2006.

[19] P. Zhang, "Programmable-logic and application-specific integrated circuits (Plasic)," in *Advanced Industrial Control Technology*, P. Zhang, Ed. Oxford: William Andrew Publishing, 2010, ch. 6, pp. 215–253. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9781437778076100063

[20] R. C. Cofer and B. F. Harding, *Rapid System Prototyping with FPGAs: Accelerating the Design Process*. Newton, MA, USA: Newnes, 2005.

[21] R. Lysecky and F. Vahid, "A study of the speedups and competitiveness of FPGA soft processor cores using dynamic hardware/software partitioning," in *Design, Automation and Test in Europe*, March 2005, vol. 1, pp. 18–23.

[22] "Picoblaze 8-bit Microcontroller," Xilinx, 2019. [Online]. Available: https://www.xilinx.com/products/intellectual-property/picoblaze.htm (accessed Mar. 4, 2021).

[23] "Lattice Mico8 Open, Free Soft Microcontroller," Lattice, 2019. [Online]. Available: http://www.latticesemi.com/Products/DesignSoftwareAndIP/IntellectualProperty/IPCore/IPCores02/Mico8.aspx

[24] "Navré avr clone (8-bit risc)," OpenCores,2019. [Online]. Available: https://opencores.org/projects/navre (accessed Mar. 4, 2021).

[25] "pavr," OpenCores, 2019. [Online]. Available: https://opencores.org/projects/pavr (accessed Mar. 4, 2021).

[26] "MCL86, MCL51, and MCL65," MicroCore Labs, 2019. [Online]. Available: http://www.microcorelabs.com/home.html (accessed Mar. 4, 2021).

[27] J. Gomez-Cornejo, A. Zuloaga, U. Bidarte, J. Jimenez, and U. Kretzschmar, "Interface tasks oriented 8-bit soft-core processor," in *Proceedings of the Annual FPGA Conference, ser. FPGAworld '12*. New York, NY, USA, ACM, 2012, pp. 4:1–4:5. [Online]. Available: http://doi.acm.org/10.1145/2451636.2451640 (accessed Mar. 4, 2021).

[28] A. Zavala, O. Camacho, J. Huerta-Ruelas, and A. Carvallo-Domínguez, "Design of a general purpose 8-bit risc processor for computer architecture learning," *Computación y Sistemas*, vol. 19, pp. 371–385, 2015.

[29] C. Ortega-Sanchez, "Minimips: An 8-bit MIPS in an FPGA for educational purposes," in *2011 International Conference on Reconfigurable Computing and FPGAs*, Nov. 2011, pp. 152–157.

[30] F. Martinez Santa, W. Sáenz Rodríguez, and F. Rivera Sánchez, "8-bit softcore microprocessor with aual accumulator designed to be used in FPGA," *Tecnura*, vol. 22, pp. 40–50, 2018.

[31] "Pauloblaze," GitHub.com, 2019. [Online]. Available: https://github.com/krabo0om/pauloBlaze (accessed Mar. 4, 2021).

[32] K. Chapman, "Picoblaze for Spartan-6, Virtex-6, 7-Series, Zynq and Ultrascale Devices (KCPSM6) - Release 9," Xilinx, Sept. 2014.
[33] "Silicon Labs - 8-bit Microcontrollers (mcus),"Silicon Labs, 2019. [Online]. Available: https://www.silabs.com/products/mcu/8-bit (accessed Mar. 4, 2021).
[34] "Embedded World - Fidex IDE," 2019. [Online]. Available: https://www.fautronix.com/en/en-fidex (accessed Mar. 4, 2021).
[35] K. Chapman, "KCPSM3 8-bit micro controller for Spartan-3, Virtex-Ii and Virtex-Iipro, Rev.7," Xilinx Ltd., Oct. 2003.
[36] P. P. Chu, *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version*. John Wiley & Sons, 2008.
[37] D. Antonio-Torres, D. Villanueva-Perez, E. Sanchez-Canepa, N. Segura-Meraz, D. GarciaGarcia, D. Conchouso-Gonzalez, J. A. MirandaVergara, J. A. Gonzalez-Herrera, A. L. R. d. Ita,B. Hernandez-Rodriguez, R. C. d. l. Monteros,F. Garcia-Chavez, V. Tellez-Rojas, and A. BautistaHernandez, "A PicoBlaze-based embedded system for monitoring applications," in *2009 International Conference on Electrical, Communications, and Computers*, Feb. 2009, pp. 173–177.
[38] V. N. Ivanov, "Using a PicoBlaze processor to traffic light control," *Cybern. Inf. Technol.*, vol. 15, no. 5, pp. 131–139, Apr. 2015. [Online]. Available: https://doi.org/10.1515/cait-2015-0023 (accessed Mar. 4, 2021).
[39] P. Zaykov, "MIMD implementation with PicoBlaze microprocessor using MPI functions," in *Proceedings of the 2007 International Conference on Computer Systems and Technologies, ser. CompSysTech '07*. New York, NY, USA, ACM, 2007, pp. 4:1–4:7. [Online]. Available: http://doi.acm.org/10.1145/1330598.1330604 (accessed Mar. 4, 2021).
[40] V. Mandala, "A study of multiprocessor systems using the PicoBlaze 8-bit microcontroller implemented on field programmable gate arrays," Master's thesis, Department of Electrical Engineering, The University of Texas at Tyler, 2011.
[41] R. D. Mattson, "Evaluation of PicoBlaze and implementation of a network interface on a FPGA," 2004. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:19730/FULLTEXT01.pdf (accessed Mar. 4, 2021).
[42] L. Claudiu, S. Sebastian, and B. Cristian, "Smart sensor implemented with PicoBlaze multiprocessors technology," in *2012 IEEE 18th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, Oct. 2012, pp. 241–245.
[43] S. M. Borawake and P. G. Chilveri, "Implementation of wireless sensor network using MicroBlaze and PicoBlaze processors," in *2014 Fourth International Conference on Communication Systems and Network Technologies*, April 2014, pp. 1059–1064.
[44] H. Pham, S. Pillement, and S. J. Piestrak, "Lowoverhead fault-tolerance technique for a dynamically reconfigurable softcore processor," *IEEE Transactions on Computers*, vol. 62, no. 6, pp. 1179– 1192, Jun. 2013.
[45] M. N. Hassan and M. Benaissa, "Embedded software design of scalable low-area elliptic-curve cryptography," *IEEE Embedded Systems Letters*, vol. 1,no. 2, pp. 42–45, Aug 2009.
[46] T. Good and M. Benaissa, "Very small FPGA application-specific instruction processor for AES," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 7, pp. 1477–1486, Jul. 2006.
[47] "ISE Design Suite," 2019. [Online]. Available: https://www.xilinx.com/products/design-tools/ise-design-suite.html (accessed Mar. 4, 2021).
[48] "Vivado Design Suite – HLx Editions," 2019. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html (accessed Mar. 4, 2021).
[49] "ISE Tutorial, Using Xilinx Chipscope Proila Core with Project Navigator to Debug FPGA Applications UG750 (v14.5)," Xilinx,Mar 2013. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_6/ug750.pdf (accessed Mar. 4, 2021).
[50] "Xilinx Zynq Ultrascale+ MPSoC ZCU104 Evaluation Kit," Xilinx, September 30, 2014. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/zcu104.html (accessed Mar. 4, 2021).
[51] "Open PicoBlaze Assembler," Kevin Thibedeau, 2017. [Online]. Available: https://kevinpt.github.io/opbasm/ (accessed Mar. 4, 2021).
[52] "Xilinx KCPSM6 Assembler," Xilinx, September 30, 2014. [Online]. Available: https://www.xilinx.com/products/intellectual-property/picoblaze.html#design (accessed Mar. 4, 2021).
[53] "Homepage of m6 - kpicosim," Xilinx, Oct 2 2009. [Online]. Available: https://marksix.home.xs4all.nl/kpicosim.html
[54] M. Szymaniak, "Picoblaze Simulator – GitHub project," Jul 31, 2017. [Online]. Available: https://github.com/sc0ty/picoblaze (accessed Mar. 4, 2021).
[55] "Sed, a stream editor," 2019. [Online]. Available: https://www.gnu.org/software/sed/manual/sed.html (accessed Mar. 4, 2021).
[56] "Improved Development Cycle for Picoblaze - GitHub Website - Xilinx Vivado 2018.3 Project." [Online]. Available: https://github.com/ehsan-ali-th/picoblaze_dev (accessed Mar. 4, 2021).
[57] B. H. Thacker, S. W. Doebling, F. M. Hemez, M. C. Anderson, J. E. Pepin, and E. A. Rodriguez, "Concepts of model verification and validation," Los Alamos National Laboratory, Sep. 2004.

[58] *Guide for the Verification and Validation of Computational Fluid Dynamics Simulations*, AIAA G-077-1998(2002), Sep. 2014.
[59] P. Knupp and K. Salari, *Verification of Computer Codes in Computational Science and Engineering*, 1st ed. Chapman and Hall/CRC, 2002.
[60] S.-L. Tsao and S.-Y. Lee, "Performance evaluation of inter-processor communication for an embedded heterogeneous multi-core processor," *Journal of Information Science and Engineering*, vol. 28, pp. 537–554, 2012.
[61] E. Ali and W. Pora, "Implementation and verification of IEEE-754 64-bit floating-point arithmetic library for 8-bit soft-core processors," in *2020 8th International Electrical Engineering Congress (iEECON)*, 2020, pp. 1–4.
[62] P. Yu and P. Schaumont, " Executing hardware as parallel software for picoblaze networks," in *2006 International Conference on Field Programmable Logic and Applications*, 2006, pp. 1–6.
[63] B. M. Gonultas, I. Yaman and T. T. Sari, "PicoTETRIS." GitHub.com. [Online]. Available: https://github.com/gonultasbu/PicoTETRIS (accessed Nov. 5, 2021).

Ehsan Ali was born in Tehran, Iran in 1983. He received the B.Eng. degree in computer systems from Assumption University of Thailand in 2015.

He is a full-time lecturer at the Computer Engineering Department, Assumption University of Thailand since 2015 and a Ph.D. candidate at Electrical Engineering Department, Chulalongkorn University of Thailand. His research interests include data centers, digital circuits, microprocessor design, reconfigurable computing, and compiler design. He is the author of several conference and journal papers with the latest article published in 2020 in International Journal of Embedded Systems.

Mr. Ali was the recipient of the 100th Anniversary Chulalongkorn University Fund for Doctoral Scholarship in 2015.

Wanchalerm Pora was born in Bangkok, Thailand in 1970. He received the B.Eng. and M.Eng. degrees in Electrical Engineering from Chulalongkorn University in 1992 & 1995 respectively. He received the Ph.D. degree from Imperial College, London in 2000.

He has joined the faculty of Engineering, Chulalongkorn University since 1994, and now working as assistant professor at the Department of Electrical Engineering. He has also served as a deputy head of the department. His research interests are in reconfigurable circuits, intelligent devices & systems for smart grid & healthcare. He has supervised many master and PhD graduate students and has 50 publications in local and international journals.

Dr. Pora is a recipient of the Chulalongkorn Academic Advancement into Its 2nd Century Project Fund.