Julia KOSOWSKA, Grzegorz MAZUR
INSTITUTE OF COMPUTER SCIENCE, WARSAW UNIVERSITY OF TECHNOLOGY
15/19 Nowowiejska St., 00-665 Warsaw, Poland

Software-Defined Computer with a Classic Microprocessor

Abstract

The paper describes the concept and the design process of a software-defined computer built for didactic purposes and as a proof-of-concept. The computer uses a classic microprocessor from the 1980s with all of its other components – memory, I/O, supplementary glue logic and a hardware bus monitor implemented with a contemporary mainstream microcontroller with Cortex-M family core. The concept of a software-defined computer was successfully implemented with four significantly different microprocessors. The computer may be used for illustrating the operation of a processor and for teaching the basics of low-level programming. It is also capable of running CP/M operating system. Its operation is controlled at both hardware and software level from a PC via two instances of a terminal emulator software.

Keywords: Software-Defined Computer, Emulation, ARM Cortex-M, STM32.

1. Teaching the basics of computer hardware

Our experiences with teaching the Computer Architecture and Microprocessor System Design show that the observation of processor's activity during the execution of an instruction is essential for understanding the computer operation by students. This is the reason for including two laboratory exercises with the Z80 CPU microprocessor in the Microprocessor System Design course lab exercises, which is otherwise based on contemporary microcomputer circuits. These hands-on exercises enable the students to watch the execution of an instruction, which is not possible with contemporary, complex and highly integrated microcomputers.

The classic microprocessor development systems built in the 1970s and 1980s are relatively big and look ancient. They also lack any interfaces that could be connected to a contemporary computer, such as USB. The idea behind our project was to build a device being a functional equivalent of those classic development kits but based on a contemporary technology, which could also be used as a teaser demonstrating the state of the art of modern microcontroller-based designs.

2. The concept of a software-defined computer

The vision of a computer using only two major components – a classic microprocessor such as the Z80 CPU and a single, complex integrated circuit – was born a few years ago. Two possibilities were considered – an FPGA-based and microcontroller-based design. While an FPGA seems to be a more natural solution for implementing basic logic circuitry for a microprocessor-based computer, finally the microcontroller version was selected for several reasons:

- It is supposed to be used as a teaser for microcontroller course and not for a programmable logic one.
- It may be easily implemented using inexpensive, off-the-shelf modules and components, affordable to every student.
- The functionality of the device may be easily modified by means of C language programming, not requiring the knowledge of hardware-description languages and FPGA synthesis tools.

It was assumed that the microcontroller will be the only significant component of our design other than the target CPU. The target computer's memory and all its peripherals were to be implemented with microcontroller's internal resources. The design was planned as flexible and easily modifiable to incorporate any typical CPU from the 1970s. Although the initial support was planned only for the Zilog Z80 CPU and Motorola MC68008, the conceptual phase included the analysis of a few more classic microprocessors, so that they could be easily supported in future versions. The microprocessors considered included the MC6800, Intel 8085, Intel 8088/NEC V20, 65C02 and Motorola MC68000/MC68010. Finally the possibility of building an Intel 8080-based version was also evaluated to commemorate the fact that an 8080A clone named MCY7880 was the only microprocessor ever produced in Poland.

3. Known similar projects

During the implementation of our design, two similar designs were published on the web by Jac Goudsmit. Both use the WDC W65C02S with the Parallax Propeller microcontroller emulating some other parts of the computer. The Parallax Propeller is a multicore microcontroller with 8 cores called cogs and 2 KB of RAM per core. Apart from that, it also has 32 KB of shared ROM, 32 KB of shared RAM called hub memory, and 32 I/O lines. Except for some hobby projects, it is not widely used.

The first of the mentioned designs, named Propeddle [1], uses a separate SRAM chip connected directly to the target processor, as well as some glue logic chips, such as latches and buffers, to compensate for the low number of I/O lines available in the Propeller. The Propeller microcontroller generates clock and control signals for the 65C02, activates the SRAM chip at the proper times and emulates I/O devices for the target processor.

The second design, named L-Star [2], is a simplified version of the Propeddle, in which the SRAM and glue logic chips were removed leaving only the 65C02 with its address, data and R/-W lines connected directly to the Propeller. The target processor's RAM is emulated using Propeller's 32 KB hub memory. Since that amount of memory may not be enough for some applications, there is a version of L-Star called L-Star Plus with a separate SRAM chip connected to the 65C02 and controlled by the Propeller much like in the Propeddle.

There are 3 main differences between ours and these designs:

- Our design is based on a popular, standard microcontroller, not a niche device.
- It does not require external memories nor any glue logic.
- It provides comprehensive monitoring facilities for observing and controlling the operation of the target processor.

4. Design goals

The goal was to design an easy to reproduce, modifiable and inexpensive device with the following characteristics:

- Single USB cable connection to a host PC computer for power, hardware control console and target processor's terminal interface.
- Basic configuration with two chips only: target CPU and the microcontroller (logic level matching circuits for some signals may be necessary in some cases).
- Full control and monitoring of target CPU, memory and I/O from hardware control console.
- A set of basic peripherals for target CPU – console/terminal, a button switch, LEDs and an interval timer; the peripherals may be optionally serviced via interrupts, so the microcontroller should also implement interrupt controller functionality.
- Optional mass storage (diskette emulation) for a target CPU enabling the use of a simple operating system such as CP/M.

5. Meeting the timing requirements of a target microprocessor

As we wanted to implement the target bus protocol using a microcontroller, we had to identify the timing requirements of

a target microprocessor and to determine if it is possible to meet them using firmware and hardware resources of the host microcontroller. There are two main timing requirements which, in a classic computer design, are normally satisfied by simple logic circuits response times. In our design, however, we had to achieve the proper response times without the use of logic gates. The two most critical timing parameters of every processor's bus protocol are:

- Data bus turnaround time at the end of a read cycle – the time from the trailing edge of a read strobe to the deactivation of data output buffers, after which the processor may drive the bus.
- READY/WAIT signal activation time enabling the target processor to recognize the bus cycle extension request.

We planned to use a Cortex-M4 or -M7 based microcontroller from the STM32 family by ST Microelectronics [3, 4], operating at the frequency in range from 72 to 200 MHz as our host device. We have estimated the realistic externally-triggered interrupt response time of a microcontroller, including the necessary firmware overhead, at approx. 250 ns in the worst case. The time was definitely too long to provide a properly timed response in both cases described above. Fortunately, the STM microcontrollers contain DMA peripherals, which could be programmed to write data to port control registers and thus to change the state of critical signals significantly faster than it was possible with purely firmware-based method.

To turn off (tristate) the data buffer and to activate the WAIT signal we had to use DMA channels. In STM32, DMA transfer requests may be generated by various peripherals, but not by port signals. To initiate DMA transfers, we programmed timers to capture external signals and to issue DMA requests on capture events. In this scenario, the timer is not used as a timer, and we do not use the timer count value latched during edge capture. In our design several features of the STM32 timers proved to be very useful including the possibility to capture signals being a combination of multiple timer inputs (when a few signals had to be monitored to detect the need to perform time-critical actions) and generating two DMA request after a capture event (when the event indicated both the need to tristate the data bus and activate the WAIT signal).

We had to perform a detailed timing constraints evaluation to determine the exact values of critical timing parameters and thus validate the possibility of building a correctly operating computer using the scheme presented. The timing constraints expressed in terms of the processor's clock cycle time were calculated using the timing requirements of the Z80 CPU and the MC68008 taken from the respective datasheets [5, 6]. In the case of most processors, including the Z80 CPU and the MC68008, both the deactivation of the data bus and activation of the bus cycle extension signal can be done after the current transfer is finished. For the Z80 CPU the trailing edge of the memory or I/O access request strobe was used as an end of transfer indicator and a trigger for both DMA transfers. For the MC68008 the trailing edge of the data strobe (indicating the validity of data on data lines in write cycles and the processor's readiness to accept data in read cycles) was used. Denoting clock cycle time as T_{CLK}, the maximum time from the trailing edge of the relevant signal to the deactivation of the data bus was ascertained as T_{CLK} − 85 ns in the case of the Z80 CPU and T_{CLK} − 30 ns in the case of the MC68008, and the maximum time from the trailing edge of the relevant signal to the activation of the bus cycle extension signal as 2 T_{CLK} − 155 ns in the case of the Z80 CPU and 2.5 T_{CLK} − 60 ns in the case of the MC68008.

Unfortunately, the DMA response time is not specified in the datasheets of STM32 microcontrollers. In order to determine if it was possible to meet the timing requirements using timers, we had to measure the DMA and port response times in a few cases, including single request and two requests issued simultaneously based on a combination of timer inputs. In order to do that, one of the timers available in STM32 microcontrollers was programmed to generate a PWM waveform with a known frequency. This signal was connected to two input pins feeding two channels of another timer that captured the rising and falling edges of the signal and generated two DMA requests of different priorities that toggled two output lines in accordance with the PWM waveform. The time intervals between the edge of the PWM waveform and the corresponding edges of the output signals were measured using an oscilloscope. The experiments performed using STM32L476RGT microcontroller running at 80 MHz together with the previously calculated timing requirements yield the maximum frequency of 4.4 MHz for the Z80 CPU and 5.8 MHz for the MC68008. Both the frequencies are higher than the minimum operating frequencies of the processors – 250 kHz and 2 MHz respectively, thus confirming the possibility of using timers and a DMA module of the STM32L4 or a higher-performance microcontroller for servicing time critical events.

6. Logic level compatibility

The STM32 microcontrollers are powered with 3.3 volts, while the classic microprocessors considered for our project work at 5 V. We had to check and adopt the signal's voltage levels to ensure the correct operation of our computer. There were several cases to be considered:

- For TTL-compatible microprocessor's inputs controlled by microcontroller's outputs the logic levels are compatible.
- For non-TTL compatible microprocessor's inputs controlled by microcontroller's outputs (such as the Z80 CPU clock input) a logic level translator IC was considered, but our measurements showed that it is enough to setup the microcontroller's output as open-drain and pull it up to 5 V supply with a 1k5 resistor.
- For CPU outputs connected to a 5 V-tolerant microcontroller inputs, direct connection with no extra components is possible.
- For a CPU outputs connected to a 3.3 V microcontroller inputs with current injection allowed only a series resistor is required.
- For processor outputs connected to a 3.3 V microcontroller inputs with no current injection allowed the simplest solution was to use a Schottky diode to convert the microprocessor output into a simulated open-drain circuit, and a pull-up resistor at the microcontroller input.

7. The implementation

In order to emulate the environment of a target processor, the microcontroller must perform the following actions:

- Detect the start of a new cycle.
- Determine the type of the transfer.
- React accordingly: if the processor is reading data, set the correct value on the data lines and activate the data output buffers; if the processor is writing data, save the data and interpret it.
- Set the READY/WAIT to let the CPU finish the current transfer.
- Once the transfer cycle is finished, deactivate the output buffers and set the READY/WAIT signal to request the extension of the next cycle giving the microcontroller ample time to service it.

Having ensured the time-critical events are serviced accordingly in hardware, the other actions required from the microcontroller emulating the environment of a processor may be implemented in software. Upon recognizing the start of a new machine cycle, an interrupt should be signaled. In the interrupt service routine, the microcontroller should determine the type of transfer requested by the processor and the address being referenced, and react accordingly. While the cycle extension request is active, the monitor can present the information about the current cycle to the user, who can decide when to let the processor finish the transfer and move to the next one. It is important to ensure that one cycle gets recognized and serviced exactly once. It requires both choosing the right set of signals to generate interrupt requests and ignoring the interrupt in some cases when it might be signaled more than once at the beginning of the same cycle.

While the first experiments were carried out using a high-performance STM32F7-series MPU, we decided to use mainstream, inexpensive STM32L476RGT microcontroller

running at 80 MHz for the final implementation due to low price and wide availability of Nucleo-64 boards equipped with this microcontroller. One of our computers is shown in Fig. 1.

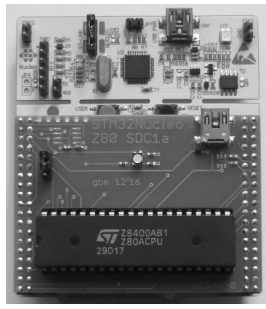

Fig. 1. Software-Defined Computer with the Z80 CPU and STML476

8. Hardware monitor

The bus monitor supports monitoring and controlling the operation of a target computer at hardware level, which includes:
- setting the frequency of the target microprocessor's clock signal;
- initializing the target (RESET signal generation);
- loading programs to the target computer's memory;
- display and modification of memory and I/O port content;
- control of target processor's operation: single-stepping, breakpoints, trace collection and fast free run mode;
- controlling the operation of the target's mass storage.

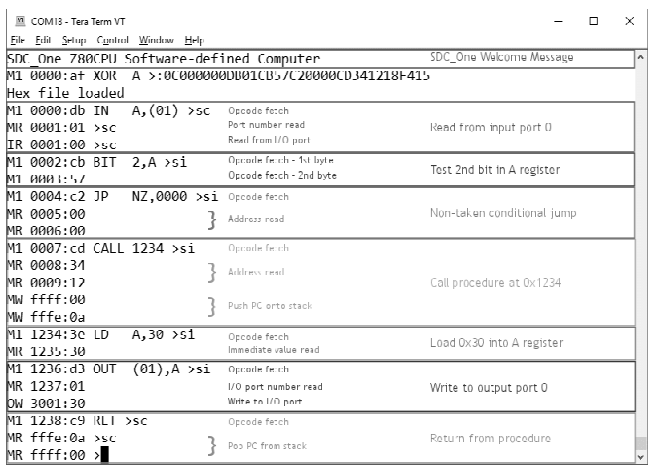

Fig. 2. Snapshot of hardware monitor session with comments in color

In cycle stepping mode the processor is suspended in each bus cycle and the information about that cycle, including its type, the address and data being transferred, is displayed on the hardware control console. In instruction stepping mode, the processor is stopped only in the cycles that indicate the beginning of a new instruction. Most implementations of the monitor support the disassembly of target CPU instructions, making it easier to monitor the actions of the processor. When the CPU is stopped, it is possible to alter its environment through the monitor commands with the effects being immediately visible and reflected in the state of data lines and the disassembled instruction display. In this way one can easily examine how particular changes affect the operation of the processor. The computer may also be put in free run mode with or without hardware breakpoints, the former enabling the maximum speed operation. Breakpoints may be defined by setting the address and the types of machine cycles in which the processor should be stopped. While in run mode the computer may collect the execution trace that may be displayed once it is stopped. Breakpoints checking and trace collection may be turned off for faster operation of the computer.

9. Conclusions

We have already successfully implemented the idea of a software defined computer using 4 different microprocessors: the Z80 CPU, Intel 8085, Motorola MC68008 and WDC W65C02S, with support for a few other processors being planned in the future. The device has been used in classes to show some aspects of the operation of a processor. We have observed some interesting features of particular processors that would be impossible to notice without the ability to follow individual machine cycles of a processor. Those included, among other things, the difference between the Intel 8085 and the Z80 CPU in the execution of a jump instruction, with the Intel 8085 optimizing away fetching a part of a target address when the jump is not taken as well as the prefetch feature of the MC68008 with the prefetched word being discarded in some cases e.g. following the change of processor's privilege level. We have measured the effective performance of our design. In the case of the Z80 CPU running at 2.5 MHz and executing a stream of NOPs the result was 357 thousand machine cycles per second (7 clock cycles per transfer), which is 1.75 times slower than best-case performance. A popular operating system from the 1980s, CP/M, has been successfully ported to the computers equipped with processors supported by the available versions of CP/M, that is CP/M-80 for the Z80 CPU and the Intel 8085, and CP/M-68k for the MC68008. A few programs have been installed including a text editor and an assembler, a source program file has been created, assembled and run successfully on the target processor thereby confirming the possibility of using our design as a fully functional computer.

10. References

[1] Goudsmit J.: Propeddle – Software Defined 6502 Computer. https://propeddle.com, 2012.
[2] Goudsmit J.: L-Star: Software-Defined 6502 Computer. https://hackaday.io/project/3620, 2014.
[3] STMicroelectronics: RM0351 STM32L4x5 and STM32L4x6 advanced ARM®-based 32-bit MCUs Reference Manual. 2016.
[4] STMicroelectronics: STM32L476xx Datasheet. 2015.
[5] Zilog: PS0178 Z8400/Z84C00 Z80 CPU Product Specification. 2002.
[6] Motorola: MC68000UM M68000 8-16-32-Bit Microprocessors User's Manual (REV 9.1). 2006.

Received: 06.02.2017 *Paper reviewed* *Accepted: 04.04.2017*

Grzegorz MAZUR, MSc

Senior Lecturer at the Institute of Computer Science, Warsaw University of Technology. His interests include computer architecture and embedded systems design.

e-mail: g.mazur@ii.pw.edu.pl

Julia KOSOWSKA

MSc student at Institute of Computer Science, Warsaw University of Technology. She received her BSc at Institute of Computer Science, Warsaw University of Technology in 2017.

e-mail: juliakosowska0@gmail.com