Thesis

HTMega
Custom Microcontroller on FPGA

Performed by	Year/Grade	Supervisor
Tobias Haubenwallner	2021/2022 5BHET	Prof. DI Christoph Wurzinger

Date: Signature:

Declaration of Honor

I hereby declare, that I wrote the present thesis for myself and without foreign help.
Furthermore, I have not used sources and aid other than those stated.

Date: Signature:

Abstract

The goal of this thesis is to develop and create a custom and functional microcontroller on an FPGA.

Functionality & Features

Core Device Functionalities

- Functional 16-bit 6 MHz computation machine consisting of CPU and memory in harvard architecture
- Implemented Turing complete instruction set

Additional Device Features

- Bootloader enabling programming over a USB connector
- 46 3.3V digital input-output pins
- Onboard PMOD connector, button, green LED and RGB LED
- USB-UART interface
- 2 general purpose 16-bit timer/counter with waveform generation and prescaler capability
- 32-bit microseconds counter

Software

- Programmer for sending assembled instructions to the device
- Assembler to compile assembly scripts
- Code highlighter for VS Code to highlight mnemonics, literals and symbols

Use Case

The developed device should be usable like a hobbyist microcontroller for use cases such as:

- Digital output, PWM output: controlling LEDs, motors, etc.
- Reading digital sensors
- Interacting with peripherals over i2c, spi, etc.
- Timing applications: clocks, repeating applications, etc.

Contents

1 Design Decisions

1.1 Preamble

After learning about logic gates in first grade, I have always wanted to build a fully functional cpu architecture. Together with going through FPGA development in electronics and learning about the ATMega328p in informatics, making a custom microcontroller seemed like a perfect thesis to me.

1.2 Architecture

Since we have gone through the ATMega328p architecture [2] in school, my design was heavily inspired by it. I wanted to improve the ATMega328p in two ways: a 16 bit architecture for bigger values and a hardware divider.

1.3 Instruction set

Similar to the architecture, the instruction set was heavily inspired by the AVR instruction set [7].
These are a few central differences:

- Changed naming convention, such as "jump" instead of "branch" (RJEQ instead of BREQ)
- Additional division
- Additional naming consistency, providing multiple variants for every instruction
- Additional jumping by flags (e.g. JFZC: Jump if Flag Zero is Set)
- No word operations
- No extended memory addressing
- No watchdog or sleep instructions
- Additional END instruction to end execution and start boot mode

1.4 Board

After going through a selection of a few FPGA development boards from producers such as Digilent, Llchitry, Terasic etc. and taking into considerations the following criteria:

- Availability
- Board & FPGA features
- Toolchain & programming effort
- Documentation
- Price

The choice fell on the Digilent CMOD-A7 35T [9] with a Xilinx Artix-7-35T FPGA [1], developed with the Vivado ML IDE [12]. The CMOD-A7 has, compared to most other development boards, little onboard peripherals which reduces complexity.
It only has microcontroller essentials such as:

- USB connector & UART driver
- Breadboardable GPIO pins
- Onboard buttons and LEDs
- Onboard RAM and flash memory

1.5 Limitations and Drawbacks

These are drawbacks that were made from the original plans:

- Reduced clock cycle from 24 MHz to 6 MHz due to timing limitations because of memory access and ALU operations
- No pipelining to reduce complexity
- Volatile internal block RAM program memory due not being able to access the onboard flash for user data
- No analog capability due to not being able to access the XADC
- The device has to be stopped to be programmed, due to not being able to access the DTR of the USB connector
- No interrupt capability to reduce complexity
- Only half of the SRAM is accessible (17 bit address) to reduce complexity

2 Manual & Datasheet

2.1 Board

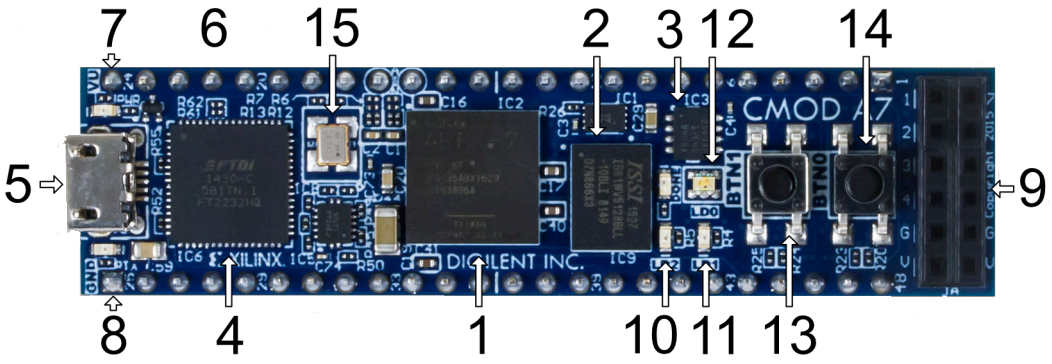

Figure 1: CMOD A7 Board Description [10]

Nr	Description
1	Artix-7 FPGA
2	512 kB SRAM (only 256 kB usable)
3	4 MB FLash (in use for FPGA configuration)
4	USB-UART Driver
5	Micro USB Connector
6	46 GPIO Pins
7	3.3V
8	GND
9	PMOD Connector
10	LED
11	Status LED
12	RGB LED
13	Button
14	Reset Button
15	12 MHz Quartz Oscillator

2.1.1 MCU Status

The Status LED is on, when the MCU is halted (in boot mode) and can be programmed.
Pressing the reset button while the MCU is running restarts the program.
Holding the reset button for 2 seconds will bring the MCU into boot mode.
When the program ends, the MCU goes into boot mode.

2.2 FPGA Block Diagram

Section inspired by [3]

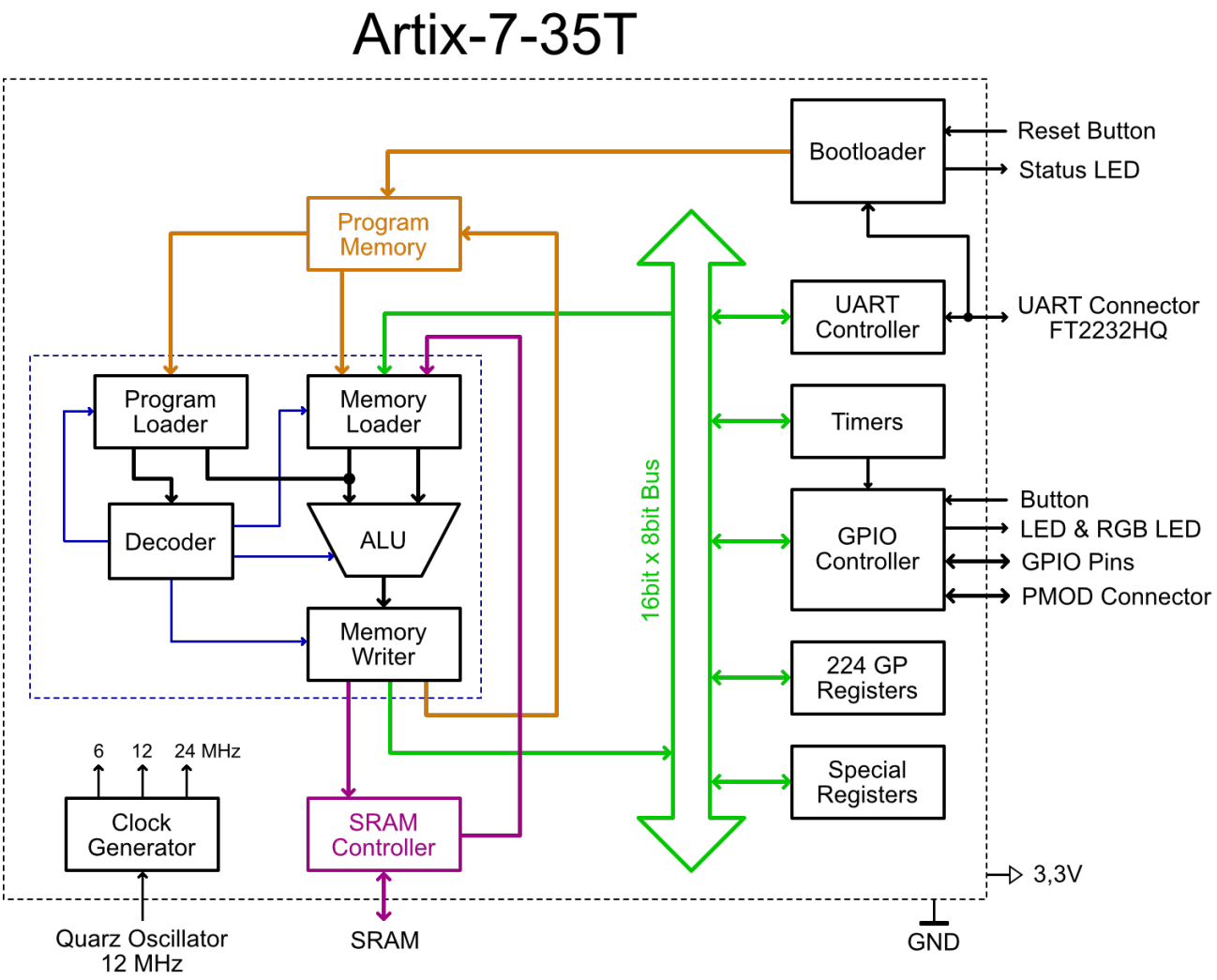

Figure 2: FPGA Design Block Diagram

The HTMega can be categorized into 2 main parts:

- the central processing unit
- the memory, split into:
  - a 8bit address, 16bit data general purpose bus
  - a 200 kB internal block memory
  - a 256 kB external SRAM

Additionally, 6, 12 and 24 MHz internal clock lines are synthesized with a MMCM analog oscillator circuit.
A Bootloader circuit is used to write to Program Memory when the MCU is in boot mode.

2.3 Memory

Section inspired by [4]

2.3.1 General Purpose Bus

The GP Bus is used to access special internal registers, 224 general purpose registers and peripherals such as UART, Timers and GPIO Pins.
Most instructions use these registers as input and/or output. (E.g. ADD, AND, CP, MOV, LDI)
There are currently 2 unused addresses at 0xE0 and 0xE1.
A list of register addresses and symbols can be found in Section 4 - Register Symbols.

2.3.2 Program Memory

The program memory is a 100k x 16bit Block Memory. One instruction has 32 bit, resulting in memory space for 50 000 instructions.
Block RAM is volatile, thus the HTMega has to be reprogrammed once power is lost.

The Program Memory can be accessed in program with instructions such as LPM and SPM.
The Program Counter addresses 32bit at once, thus the program counter can access the address range of 100k. To access the whole address range in instructions, 17 bits are needed. The 17th bit is represented by the P flag.

2.3.3 SRAM

The CMOD A7 board has a Macronix MX25L3233F 512kB SRAM on board,
256 kB of which are usable.
To address the 128k x 16bit, a 17th address bit is needed, represented by the R flag.

The SRAM is accessed with instructions such as LD and STM.

Instructions such as PUSH and POP use the Stack Pointer register as SRAM address
and increment or decrement the Stack Pointer.

2.4 Execution Cycle

Execution happens at a 6 MHz clock cycle and the execution length depends on the instruction.
Execution follows this general cycle, depending on the instruction omitting certain operations:

> load instruction - decode instruction - load from memory - arithmetic operation - write to memory

Example

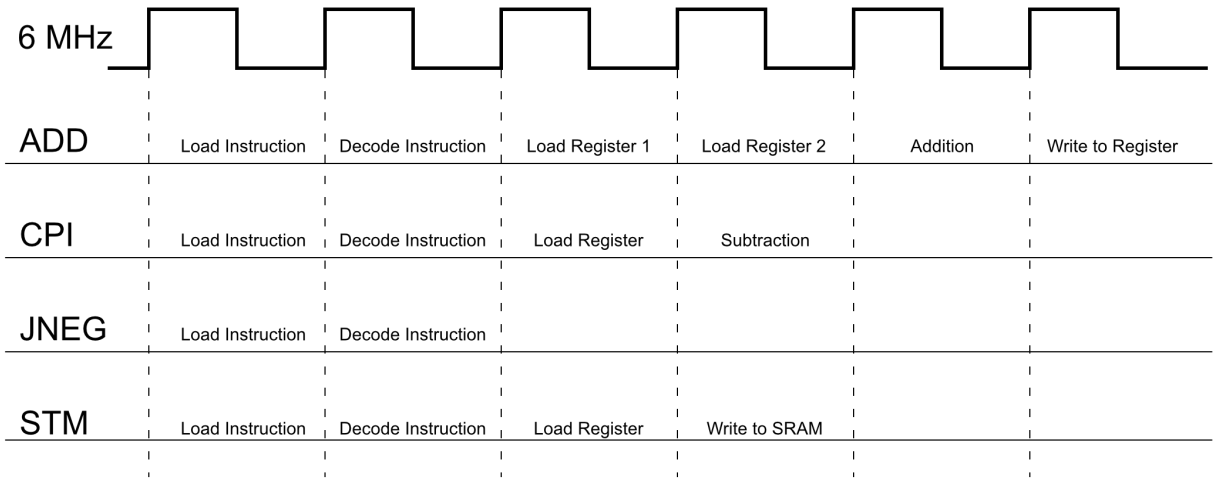

Figure 3: Example Execution Cycle

2.5 Programming the Microcontroller

The program is saved on an internal RAM and thus volatile.
Once disconnected, the HTMega will have to be reprogrammed.

2.5.1 Assembly

Assembly code consists of consecutive instructions.
All instructions for the HTMega can be found in Section 3 - Instruction Set

Every instruction uses 4 bytes of program memory.
Labels are yet to be supported, thus you will have to set any jump addresses yourself. Since every instruction has the same length, jump to the instruction line number - 1 (zero-based numbering).

Every program ends with an END instruction, if it is missing, the assembler will add it automatically.
Any instruction after END will be ignored.

Special and IO registers can additionally be addressed with symbols.
A list of all Symbols can be found in Section 4 - Register Symbols.
Literal numbers are formatted as binary with 0b and hexadecimal with 0x.
Inline comments start with //.

Example Code

```
PUSH PC 0b1011 // push Program Counter to sram address 0b1011
JMP 0x00 // jump to instruction 0
END
```

An extension for HTMega assembly code highlighting is available for Visual Studio Code and can be installed through a .VSIX file included with the binaries.

Assembling

To assemble the script, pass the path of the script to the HTMega assembler:

```
> HTMegaAssembler.exe <input.hasm> <output.hex>
```

Accepted file types are .hasm and .asm for input, .hex and .bin for output. Omitting the output path will create a output hex file at the same path with the same name as the input.

2.5.2 Programmer

To program the HTMega, (re)connect the board over USB or hold the reset button to halt the program. Then pass the .hex file created by the assembler to HTMegaProgrammer.exe:

```
> HTMegaProgrammer.exe <input.hex> <COM-Port>
```

If no COM-Port is specified, a list of available COM-Ports will be shown to select from.

If the COM-Port exists, is not occupied and the HTMega is in boot mode, the assembled binary code is sent to the HTMega and execution is started automatically.

Programming Protocol

When the MCU is in boot mode, the Bootloader is listening for the start-byte 0xF0, followed by the assembled instructions. The transmission ends with 6x 0xFF.

2.6 Pinout

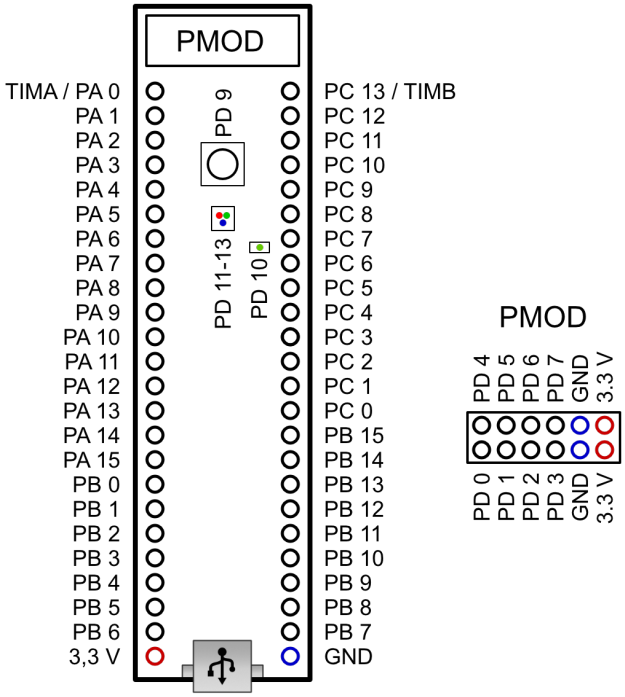

Figure 4: Pinout [8]

2.7 GPIO

Section inspired by [5]

As seen in the pinout above, the pins and IOs are split into 4 register groups.
Port A-C control the breadboardable pins, Port D controls onboard IO like LEDs and buttons,
as well as the PMOD connector.

All pins are digital and can take the states 3.3V or 0V, matching 1 or 0 in the registers.
Each group consists of three registers: input, output and output enable. (Px_in, Px_out, Px_conf)

The input register matches the state the pins have. The output register sets the state of a pin.
All GPIO pins and PMOD pins are set to high impedance by default,
so when connecting a voltage from the outside, there won't be a short circuit.
To connect a bit in the output register to the assigned pin, the associated bit in the output enable register has to be 1.

The onboard LED (PD 10) and RGB LED (PD 11 = red, PD 12 = green, PD 13 = blue) are always connected to the output register. Output enable has no effect on these pins.
The onboard button (PD 9) is never connected to the output register. When output enable is set to 0 for pins PA 0 and PC 13, they are connected to Timer A and Timer B.

Block Diagram

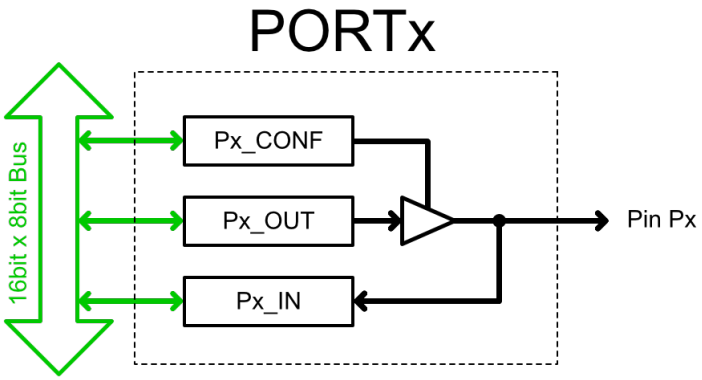

Figure 5: GPIO Interface Block Diagram

Example Code

```
LDI pa_conf 0b100 // enable PA 2 output
LDI pa_out 0b100  // set PA 2 to 1
XORI pd_out 0x200 // toggle onboard LED
END
```

2.8 USB - UART

The CMOD A7 board has a FT2232HQ USB-UART converter enabling UART communication over UART for the HTMega.

2.8.1 Protocol

UART protocol as found in [13] A UART frame used by the HTMega consists of: 1 low start bit, 8 data bits, an even or odd parity bit, 2 high stop bits. The max baud rate is 6 MHz and can be reduced by the baud rate divider.
12 bits with a maximum baud rate of 6 MHz results to a maximum data transfer rate of 500 kB/s

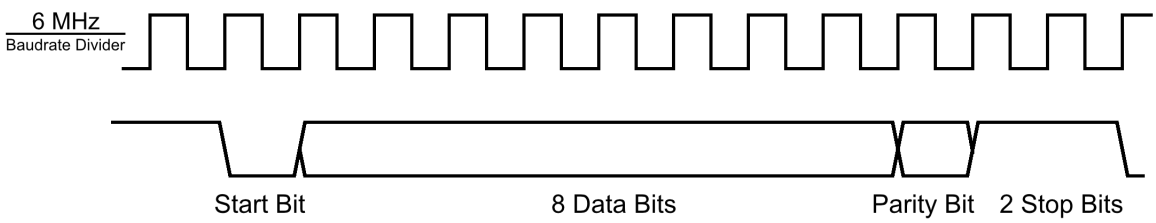

Figure 6: UART Frame

2.8.2 Configuration

UART configuration is stored in the register UART_CONF.

15 - 2	1	0
Baud Rate Divider	Parity	Enable

Enable	Enable Sending and Receiving
Parity	Even parity (0), Odd parity (1)
Baud Rate Divider	Set UART Baud rate

$$UART\ Baud\ Rate = \frac{6MHz}{Baud\ Rate\ Divider} \tag{1}$$

If the Baud Rate Divider is 0, the Baud rate is set to 6 MHz.

2.8.3 Sending

Transmission data is stored in the Register UART_TD. Enable has to set to 1 to enable transmission. Writing to UART_TD starts the transmission and the lower byte of UART_TD will be transmitted. While the transmission is active, the 'sending' bit in UART_RD is 1.

2.8.4 Receiving

UART status and received data are stored UART_RD.
Once enabled, the UART controller is always listening for UART frames coming over USB.
While a frame is being received, 'receiving' in UART_RD is 1.
Once the frame is fully received, 'received' is set to 1 and the received data is transferred to 'Received Data' in UART_RD. If the parity of the received frame does not match the set parity (even by default), 'wrong parity' is set to 1.

UART_RD Register Description

15 - 12	11	10	9	8	7 - 0
x	wrong parity	received	receiving	sending	Received Data

Received Data	Data of the last received UART frame
sending	Status: is currently sending
receiving	Status: is currently receiving
received	Status: finished receiving current UART frame
wrong parity	Status: wrong parity in the last received UART frame

Writing to UART_RD won't overwrite any data, but will reset 'wrong parity', 'received' and 'Received Data'.

2.9 Timers

Section inspired by [6]

The HTMega has 2 universal 16 bit timers/counters and a 32 bit microseconds counter.

2.9.1 Universal Timer/Counter

The Universal 16bit Timer/Counter is a binary counter with a configurable top value,
a comparison unit and the ability to generate PWM on a certain pin.
For the actual position of the IO pin, refer to section 2.6 - Pinout.

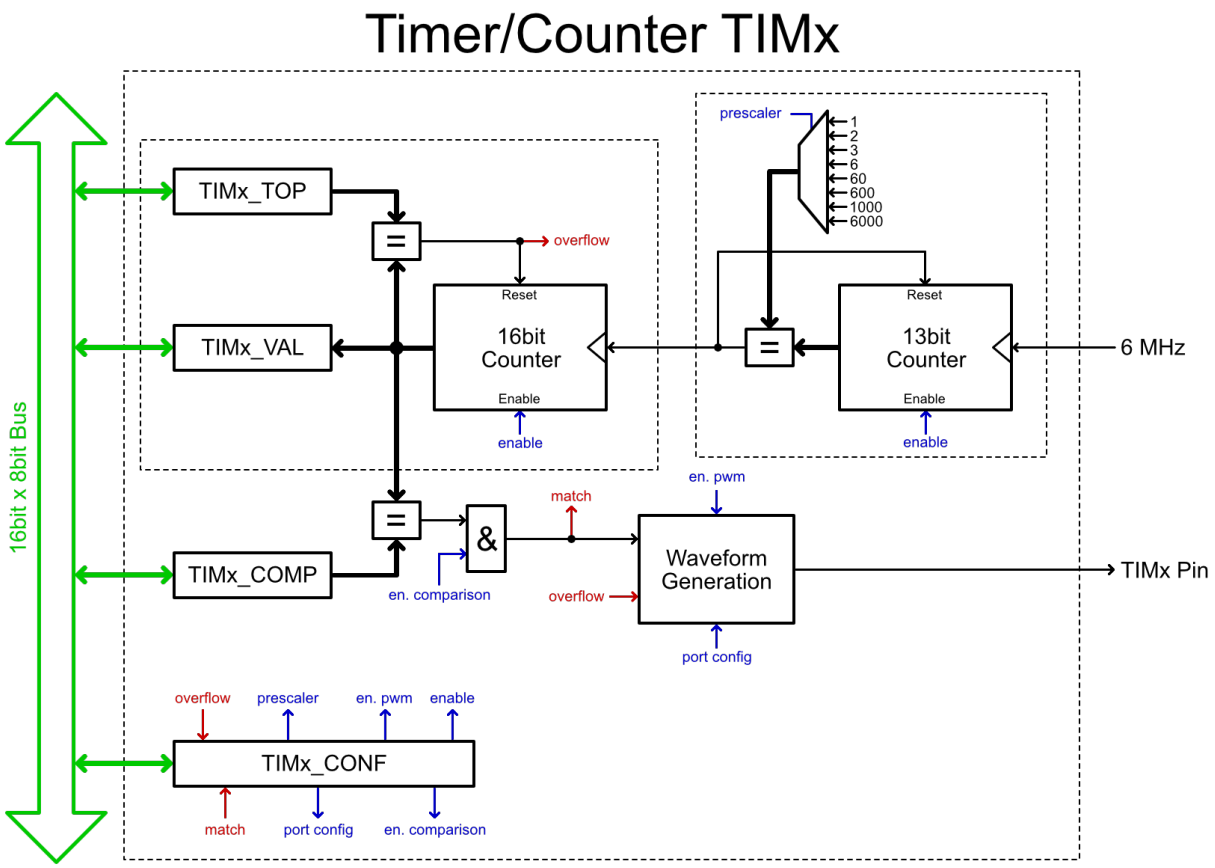

Figure 7: Universal Timer/Counter Block Diagram

2.9.2 Configuration

Timer/Counter configuration and status are stored in TIMx_CONF.

15 - 10	9	8	7 - 5	4 - 3	2	1	0
x	match	overflow	prescaler	port config	en. pwm	en. comp.	enable

match	Status: counter value matched comparison value
overflow	Status: counter reached top
Prescaler	select prescaler value
Port Config	configure waveform generation
en. pwm	enable waveform generation
en. comp	enable comparison
enable	enable Timer/Counter

Prescaler

The prescaler reduces the counter input clock frequency.
A prescaler value can be selected from a preset of 8 values.

Prescaler	000	001	010	011	100	101	110	111
Divider	1	2	3	6	60	600	1000	6000
Frequency	6 MHz	3 MHz	2 MHz	1 MHz	100 kHz	10 kHz	5 kHz	1 kHz

Waveform Generation

Port Config configures the behaviour of the assigned port.

en. pwm = 0

Port Config	Operation
00	none
01	toggle port on match
10	reset port on match
11	set port on match

en. pwm = 1

Port Config	Operation
00	none
01	toggle port on match
10	fast PWM: set port on zero, reset port on match
11	reverse PWM: reset port on zero, set port on match

3 Instruction Set

3.1 Nomenclature

Registers

PC	Program Counter
SP	Stack Pointer
B	Secondary ALU Output
Z	Indirect Adress Register

Flags

C	Carry
Z	Zero
N	Negative
V	Two's Complement overflow
I	Global Interrupt enable
T	Bit Storage
R	Ram Address Extension
P	PM Address Extension

Operands

Rd	(8bit) Write Register Address
Rr	(8bit) Read Register Address
K	(16bit) Immediate Value
X	(16bit) RAM/PM Address
k	(16bit) Branch Address

Nomenclature

(x)	Access to Memory over Address x
Reg	General purpose Register
Imdt	Immediate Value
r	Division remainder
[u]	unsigned
[s]	signed

3.2 Arithmetic & Logic Instructions

Mnemonic	Operands	Operation	Description	Flags	Cycles
ADDC	Rd, Rr	(Rd) ← (Rd) + (Rr) + C	Add with Carry	Z, C, N, V	6
ADD	Rd, Rr	(Rd) ← (Rd) + (Rr)	Add Registers	Z, C, N, V	6
ADDI	Rd, K	(Rd) ← (Rd) + K	Add Imdt	Z, C, N, V	5
ADDCI	Rd, K	(Rd) ← (Rd) + K + C	Add Imdt and Carry	Z, C, N, V	5
SUB	Rd, Rr	(Rd) ← (Rd) - (Rr)	Subtract Registers	Z, C, N, V	6
SUBC	Rd, Rr	(Rd) ← (Rd) - (Rr) - C	Subtract with Carry	Z, C, N, V	6
SUBI	Rd, K	(Rd) ← (Rd) - K	Subtract Imdt	Z, C, N, V	5
SUBCI	Rd, K	(Rd) ← (Rd) - K - C	Subtract Imdt with Carry	Z, C, N, V	5
MUL	Rd, Rr	B:(Rd) ← [u](Rd) * [u](Rd)	Unsigned Multiply Registers	Z, N	6
MULS	Rd, Rr	B:(Rd) ← [s](Rd) * [s](Rd)	Signed Multiply Registers	Z, N	6
MULI	Rd, K	B:(Rd) ← [u](Rd) * [u]K	Unsigned Multiply with Imdt	Z, N	5
MULSI	Rd, K	B:(Rd) ← [s](Rd) * [s]K	Signed Multiply with Imdt	Z, N	5
DIV	Rd, Rr	(Rd) ← [u](Rd) / [u](Rr) B ← r	Unsigned Divide Registers	Z, N	6
DIVS	Rd, Rr	(Rd) ← [s](Rd) / [s](Rr) B ← r	Signed Divide Registers	Z, N	6
DIVI	Rd, K	(Rd) ← [u](Rd) / [u]K B ← r	Unsigned Divide by Imdt	Z, N	5
DIVSI	Rd, K	(Rd) ← [s](Rd) / [s]K B ← r	Signed Divide by Imdt	Z, N	5
AND	Rd,Rr	(Rd) ← (Rd) ∧ (Rr)	Logical AND	Z, N	6
ANDI	Rd,K	(Rd) ← (Rd) ∧ K	Logical AND with Imdt	Z, N	5
OR	Rd,Rr	(Rd) ← (Rd) ∨ (Rr)	Logical OR	Z, N	6
ORI	Rd,K	(Rd) ← (Rd) ∨ K	Logical OR with Imdt	Z, N	5
XOR	Rd,Rr	(Rd) ← (Rd) ⊕ (Rr)	Logical XOR	Z, N	6
XORI	Rd,K	(Rd) ← (Rd) ⊕ K	Logical XOR with Imdt	Z, N	5

Mnemonic	Operands	Operation	Description	Flags	Cycles
CP	Rr1, Rr2	(Rr1) - (Rr2)	Compare Registers	Z, C, N, V	5
CPC	Rr1, Rr2	(Rr1) - (Rr2) - C	Compare with Carry	Z, C, N, V	5
CPI	Rr, K	(Rr) - K	Compare with Imdt	Z, C, N, V	4
CPCI	Rr, K	(Rr) - K - C	Compare with Imdt and Carry	Z, C, N, V	4
BTST	Rr1, Rr2	(Rr1) ∧ (Rr2)	Test Bit	Z, N	5
BTSTI	Rr, K	(Rr) ∧ K	Best Bit with Imdt	Z, N	4
COM	Rd	(Rd) ← 0xFFFF - (Rd)	One's Complement	Z, N	5
NEG	Rd	(Rd) ← 0x0000 - (Rd)	Two's Complement	Z, N	5
INC	Rd	(Rd) ← (Rd) + 1	Increment Register	Z, C, N, V	5
DEC	Rd	(Rd) ← (Rd) - 1	Decrement Register	Z, C, N, V	5
SWAP	Rd	(Rd)[0-7] ← (Rd)[8-15] (Rd)[8-15] ← (Rd)[0-7]	Swap Bytes	Z, N	5
LSL	Rd	(Rd) ← (Rd) << 1	Logical shift left	Z, C, N, V	5
LSR	Rd	(Rd) ← (Rd) >> 1	Logical shift right	Z, C, N, V	5
ROL	Rd	(Rd) ← (Rd) << 1 (Rd)[0] ← (Rd)[15]	Rotate left	Z, N	5
ROR	Rd	(Rd) ← (Rd) >> 1 (Rd)[15] ← (Rd)[0]	Rotate right	Z, N	5
ASR	Rd	(Rd) ← (Rd) >> 1 (Rd)[15] ← (Rd)[15]	Arithmetic right shift	Z, C, N, V	5
TST	Rd	(Rd)	Test Register	Z, N	4

3.3 Branch/Jump Instructions

Mnemonic	Operands	Operation	Description	Flags	Cycles
JMP	k	PC ← k	Jump		2
RJMP	k	PC ← PC + k + 1	Relative Jump		2
IJMP	k	PC ← Z	Indirect Jump		2
RIJMP	k	PC ← Z	Relative indirect Jump		2

Mnemonic	Operands	Operation	Description	Flags	Cycles
JEQ	k	if(Z): PC ← k	Jump if equal	Z	2
JNEQ	k	if(∼Z): PC ← k	Jump if not equal	Z	2
JHI	k	if (∼C ∧ ∼Z): PC ← k	Jump if higher	C, Z	2
JSHI	k	if (∼C): PC ← k	Jump if same or higher	C	2
JSLO	k	if (C ∨ Z): PC ← k	Jump if same or lower	C, Z	2
JLO	k	if (C): PC ← k	Jump if lower	C	2
JMI	k	if (N): PC ← k	Jump if minus	N	2
JPL	k	if (∼N): PC ← k	Jump if plus	N	2
JGE	k	if (N ⊕ ∼V): PC ← k	Jump if greater or equal, signed	N, V	2
JLT	k	if (N ⊕ V): PC ← k	Jump if less than, signed	N, V	2
RJEQ	k	if(Z): PC ← PC + k + 1	Relative Jump if equal	Z	2
RJNEQ	k	if(∼Z): PC ← PC + k + 1	Relative Jump if not equal	Z	2
RJHI	k	if (∼C ∧ ∼Z): PC ← PC + k + 1	Relative Jump if higher	C, Z	2
RJSHI	k	if (∼C): PC ← PC + k + 1	Relative Jump if same or higher	C	2
RJSLO	k	if (C ∨ Z): PC ← PC + k + 1	Relative Jump if same or lower	C, Z	2
RJLO	k	if (C): PC ← PC + k + 1	Relative Jump if lower	C	2
RJMI	k	if (N): PC ← PC + k + 1	Relative Jump if minus	N	2
RJPL	k	if (∼N): PC ← PC + k + 1	Relative Jump if plus	N	2
RJGE	k	if (N ⊕ ∼V): PC ← PC + k + 1	Relative Jump if greater or equal, signed	N, V	2
RJLT	k	if (N ⊕ V): PC ← PC + k + 1	Relative Jump if less than, signed	N, V	2

3.4 Branch by Flag Instructions

Mnemonic	Operands	Operation	Description	Flags	Cycles
JFZS	k	if (Z): PC ← k	Jump if Zero flag set	Z	2
JFCS	k	if (C): PC ← k	Jump if Carry flag set	C	2
JFNS	k	if (N): PC ← k	Jump if Negative flag set	N	2
JFVS	k	if (V): PC ← k	Jump if Overflow flag set	V	2
JFTS	k	if (T): PC ← k	Jump if Bit Storage flag set	T	2
JFIS	k	if (I): PC ← k	Jump if Interrupt flag set	I	2
JFRS	k	if (R): PC ← k	Jump if Ram Select flag set	R	2
JFPS	k	if (P): PC ← k	Jump if PM Select flag set	P	2
JFZC	k	if (∼Z): PC ← k	Jump if Zero flag cleared	Z	2
JFCC	k	if (∼C): PC ← k	Jump if Carry flag cleared	C	2
JFNC	k	if (∼N): PC ← k	Jump if Negative flag cleared	N	2
JFVC	k	if (∼V): PC ← k	Jump if Overflow flag cleared	V	2
JFTC	k	if (∼T): PC ← k	Jump if Bit Storage flag cleared	T	2
JFIC	k	if (∼I): PC ← k	Jump if Interrupt flag cleared	I	2
JFRC	k	if (∼R): PC ← k	Jump if Ram Select flag cleared	R	2
JFPC	k	if (∼P): PC ← k	Jump if PM Select flag cleared	P	2
RJFZS	k	if (Z): PC ← PC + k + 1	Relative Jump if Zero flag set	Z	2
RJFCS	k	if (C): PC ← PC + k + 1	Relative Jump if Carry flag set	C	2
RJFNS	k	if (N): PC ← PC + k + 1	Relative Jump if Negative flag set	N	2
RJFVS	k	if (V): PC ← PC + k + 1	Relative Jump if Overflow flag set	V	2
RJFTS	k	if (T): PC ← PC + k + 1	Relative Jump if Bit Storage flag set	T	2
RJFIS	k	if (I): PC ← PC + k + 1	Relative Jump if Interrupt flag set	I	2
RJFRS	k	if (R): PC ← PC + k + 1	Relative Jump if Ram Select flag set	R	2

Mnemonic	Operands	Operation	Description	Flags	Cycles
RJFPS	k	if (P): PC ← PC + k + 1	Relative Jump if PM Select flag set	P	2
RJFZC	k	if (∼Z): PC ← PC + k + 1	Relative Jump if Zero flag cleared	Z	2
RJFCC	k	if (∼C): PC ← PC + k + 1	Relative Jump if Carry flag cleared	C	2
RJFNC	k	if (∼N): PC ← PC + k + 1	Relative Jump if Negative flag cleared	N	2
RJFVC	k	if (∼V): PC ← PC + k + 1	Relative Jump if Overflow flag cleared	V	2
RJFTC	k	if (∼T): PC ← PC + k + 1	Relative Jump if Bit Storage flag cleared	T	2
RJFIC	k	if (∼I): PC ← PC + k + 1	Relative Jump if Interrupt flag cleared	I	2
RJFRC	k	if (∼R): PC ← PC + k + 1	Relative Jump if Ram Select flag cleared	R	2
RJFPC	k	if (∼P): PC ← PC + k + 1	Relative Jump if PM Select flag cleared	P	2
MOV	Rr, Rd	(Rd) ← (Rr)	Move Register		4
LD	Rd, X	(Rd) ← RAM(X)	Load from RAM		4
LDI	Rd, K	(Rd) ← K	Load Imdt		3
ILD	Rd	(RD) ← RAM(Z)	Indirect Load from RAM		4
ST	Rr, X	RAM(X) ← (Rr)	Store on RAM		4
IST	Rr	RAM(Z) ← (Rr)	Indirect Store on RAM		4
ISTI	K	RAM(Z) ← K	Indirect Store Imdt on RAM		3
PUSH	Rr	RAM(SP) ← (Rr) SP ← SP - 1	Push to Stack		4

Mnemonic	Operands	Operation	Description	Flags	Cycles
PUSHI	K	RAM(SP) ← K SP ← SP - 1	Push Imdt to Stack		3
POP	Rd	(Rd) ← RAM(SP) SP ← SP + 1	Pop from Stack		4
LPM	Rd, X	(Rd) ← PM(X)	Load from PM		4
ILPM	Rd	(Rd) ← PM(Z)	Indirect Load from PM		4
SPM	Rr, X	PM(X) ← (Rr)	Store on PM		4
ISPM	Rr	PM(Z) ← (Rr)	Indirect Store on PM		4
ISPMI	K	PM(Z) ← K	Indirect Store Imdt on PM		3

3.5 Control Instructions

Mnemonic	Operands	Operation	Description	Flags	Cycles
NOP			No Operation		2
END			End Program, start Boot Mode		2

4 Register Symbols

Symbol	Address	Description
ALU_B	0xFB	Alu Secondary Output
SP	0xFC	Stack Pointer
IADDR	0xFD	Indirect Address Register (Z-Register)
PC	0xFE	Program Counter (Readonly)
FLAGS	0xFF	Flag Register

Symbol	Address	Description
PA_IN	0xE2	Port A input
PA_OUT	0xE3	Port A output
PA_CONF	0xE4	Port A output enable
PB_IN	0xE5	Port B input
PB_OUT	0xE6	Port B output
PB_CONF	0xE7	Port B output enable
PC_IN	0xE8	Port C input
PC_OUT	0xE9	Port C output
PC_CONF	0xEA	Port C output enable
PD_IN	0xEB	Port D input
PD_OUT	0xEC	Port D output
PD_CONF	0xED	Port D output enable
UART_CONF	0xEE	UART config
UART_TD	0xEF	UART Transfer Data
UART_RD	0xF0	UART Received Data
TIMA_VAL	0xF1	Timer A Value (resets timer on write) (readonly)
TIMB_VAL	0xF2	Timer B Value (resets timer on write) (readonly)
TIMA_TOP	0xF3	Timer A Top Value (resets timer on write)
TIMB_TOP	0xF4	Timer B Top Value (resets timer on write)
TIMA_COMP	0xF5	Timer A Comparison Value
TIMB_COMP	0xF6	Timer B Comparison Value
TIMA_CONF	0xF7	Timer A Configuration
TIMB_CONF	0xF8	Timer B Configuration
MICR_L	0xF9	Micros Counter Lower 16bit
MICR_H	0xFA	Micros Counter Higher 16bit

5 Software

Both the Assembler and the Programmer are C# console applications developed in Visual Studio.
The code highlighter is a Node-JS Visual Studio Code Extension project developed in VS Code.

5.1 Assembler

The HTMega Assembler read HTMega instructions from a .hasm or .asm file and assembles them into bytecode inside a .hex or .bin file.

5.1.1 Running the Assembler

The Assembler has to be called over console with up to two arguments:

```
> HTMegaAssembler.exe <input .hasm/.asm> <(optional) output .hex/.bin>
```

If the second argument is omitted, the resulting output hex file will be placed next to the input file with the same name and the .hex extension.

5.1.2 Errors

The following additional errors are implemented:

- Argument Error:
  - Wrong amount of arguments
  - Argument not in path format
  - Input file does not exist
  - Wrong extension
- Syntax Error:
  - Invalid Mnemonic
  - Wrong amount of parameters
- Value Error:
  - Parameter not of type uint8 (register address), uint16 (PM/RAM address) or int16 (immediate value)
  - PM address exceeds address range of 50000

5.1.3 Flow Chart

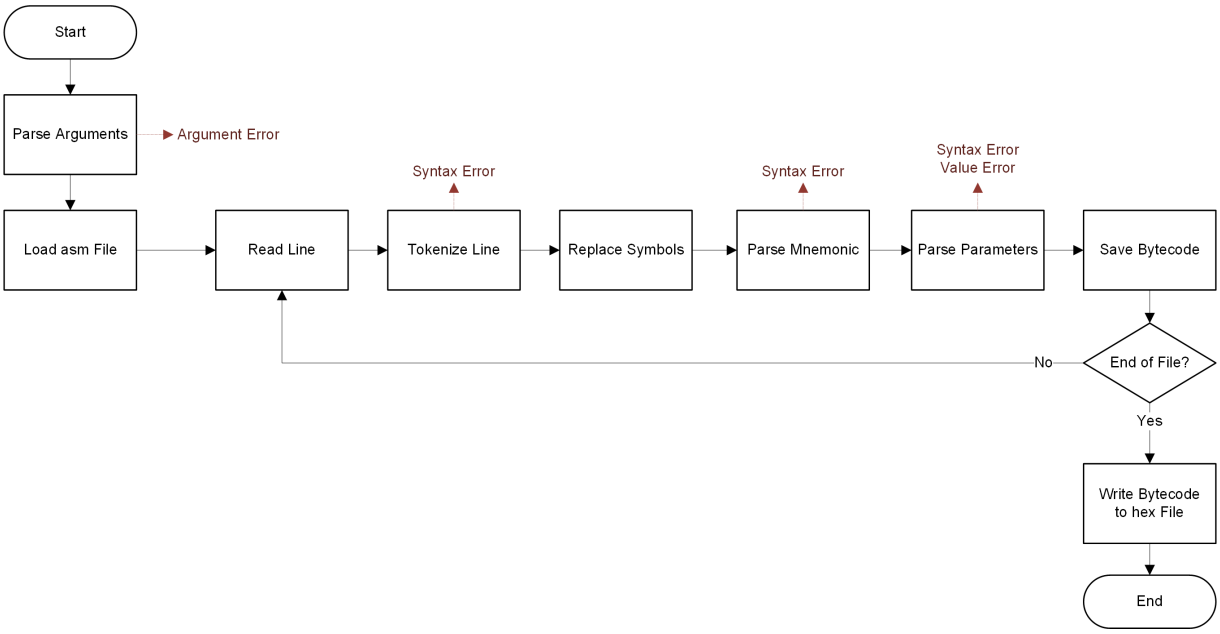

Figure 8: Assembler Flow Chart

5.2 Programmer

The HTMega Programmer reads a .hex or .bin file and sends the assembled program to the HTMega over USB.
The transmission to the HTMega begins with the start-byte 0xF1, followed by the bytecode.

5.2.1 Running the Programmer

The Programmer has to be called over console with up to two arguments:

```
> HTMegaAssembler.exe <input .hex/.bin> <(optional) COM-Port>
```

If the second Argument is omitted, a selection with available COM-Ports will be shown.

5.2.2 Errors

The following errors are implemented, next to the C# serial port exceptions [11] :

- Argument Error:
  - Wrong amount of arguments
  - Argument not in path format
  - Input file does not exist
  - Wrong extension
  - COM-Port does not exist or is not available

5.3 Code Highlighter

VS Code syntax highlighting [17] is powered by TextMate Grammar [14] and uses regular expressions to select certain text. The HTMega .hasm highlighter for consists of a tmLanguage.json file that includes the code highlighting configuration.

Example - Regular Expression to select literals:

```
(?i)\\b(([0-9]+)|(0x[0-9a-f]*)|(0b[01]*))\\b
```

A list of regular expressions can be found in [15] - *TextMate Regular Expressions*.

6 Getting Started

This section describes how to load the FPGA bitstream on an empty CMOD-A7 board.

Requirements

The following resources and pieces of software are needed:

- the HTMega-FPGA project, containing design sources and configurations
- the Vivado-ML IDE (more details in [12] - *Getting Started with Vivado*)
- the CMOD A7 board files (more details in [16] - *Vivado Board Files Installation*)

The standard version of Vivado is free to use, but requires a Xilinx account.

Generating Bitstream

Open the HTMega_FPGA.xpr file inside the project folder. Even though the project folder contains the finished bitstream, it is recommended to generate a bitstream after opening the project for the first time by clicking on 'generate bitstream' on the left panel. This will also launch synthesis and implementation. Click on 'Yes' or 'OK' when Vivado asks if you want to launch Synthesis and Implementation.
This might take, depending on your pc, 10 - 20 minutes.

Connecting to the Target

Once the bitstream generation is finished, a dialog will pop up asking if you want to open the hardware manager, click yes. If you want to open the hardware manager afterwards, click 'open hardware manager' below. After opening the hardware manager, click on 'Open Target' and 'Auto Connect' in the top bar.

Volatile Programming

Programming the Artix-7 over JTAG will only load the bitstream onto the FPGA, which is volatile. Thus, once you remove the power, the FPGA configuration will be lost. To program the FPGA over JTAG, right-click on the Artix-7 device and click on 'Program Device'.

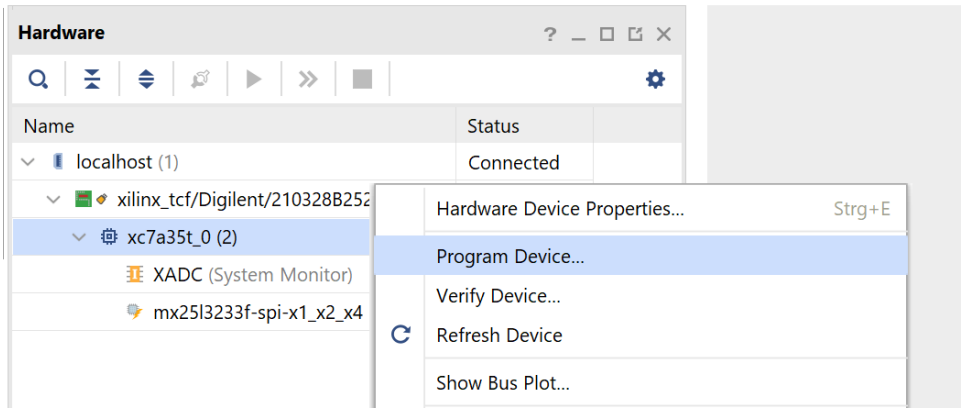

Figure 9: Programming over JTAG

This might take up to a minute.

Nonvolatile Programming

To make the FPGA configuration constant, the bitstream will have to be loaded onto the onboard flash. The Artix-7 will then load the bitstream from the flash on every startup. To load the bitstream onto the flash, right-click on the memory device and click on 'Program Configuration Memory Device'.

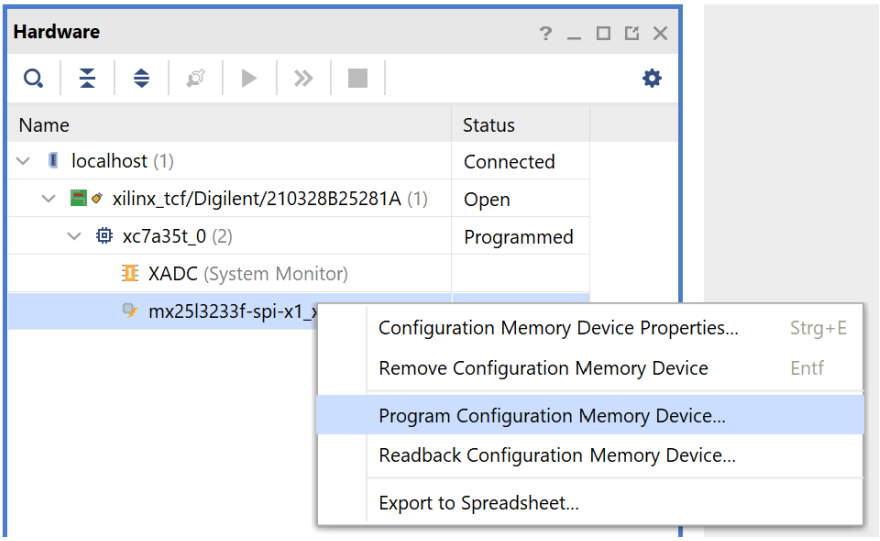

Figure 10: Programming the configuration memory device

This will take a lot longer than JTAG programming, but once the flash is programmed, every startup will only take a few milliseconds.

List of Figures

References

[1] *7Series Overview*. Xilinx. URL: https://docs.xilinx.com/v/u/en-US/ds180_7Series_Overview.

[2] *ATmega328p Datasheet*. Atmel Corporation. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[3] *ATmega328p Datasheet*. Atmel Corporation. Chap. 6. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[4] *ATmega328p Datasheet*. Atmel Corporation. Chap. 7. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[5] *ATmega328p Datasheet*. Atmel Corporation. Chap. 13. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[6] *ATmega328p Datasheet*. Atmel Corporation. Chap. 14. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[7] *AVR Instruction Set Manual*. Atmel Corporation. Chap. 4. URL: http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf.

[8] *CMOD A7 pinout*. Digilent. URL: https://digilent.com/reference/_media/cmod_a7/cmoda7_b_dip.png.

[9] *CMOD A7 Reference Manual*. Digilent. URL: https://digilent.com/reference/_media/reference/programmable-logic/cmod-a7/cmod_a7_rm.pdf.

[10] *CMOD A7 top-down image*. Digilent. URL: https://digilent.com/reference/_media/reference/programmable-logic/cmod-a7/cmod-a7-1.png.

[11] *CSharp Open Serial Port*. Microsoft. URL: https://docs.microsoft.com/en-us/dotnet/api/system.io.ports.serialport.open.

[12] *Getting Started with Vivado*. Xilinx. URL: https://www.xilinx.com/developer/products/vivado.html.

[13] Eric Pena. *UART: a hardware communication protocol*. URL: https://www.analogbbb.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html.

[14] *TextMate Language Grammars*. MacroMates Ltd. URL: https://macromates.com/manual/en/language_grammars.

[15] *TextMate Regular Expressions*. MacroMates Ltd. URL: https://macromates.com/manual/en/regular_expressions.

[16] *Vivado Board Files Installation*. Digilent. URL: https://digilent.com/reference/software/vivado/board-files.

[17] *VS Code Syntax Highlighting Guide*. Microsoft. URL: https://code.visualstudio.com/api/language-extensions/syntax-highlight-guide.

Time Tracking

Date	Hours	Description
03.07.2021	05:48	installing & testing Vivado
15.07.2021	09:40	learning HDL & testing CMOD A7
16.07.2021	09:00	starting counter & early ALU
17.07.2021	09:12	ALU & RAM Controller
27.07.2021	02:30	starting UART Controller
27.10.2021	06:20	UART Controller
28.10.2021	09:00	UART Controller
31.10.2021	03:20	UART Controller
02.11.2021	04:51	UART Controller
25.11.2021	05:34	Flash Controller
26.11.2021	06:00	Flash Controller
27.11.2021	03:00	Flash Controller
01.12.2021	07:10	Multiplier
02.12.2021	04:30	Multiplier
18.12.2021	06:00	ALU
03.01.2022	10:00	GP Bus
04.01.2022	09:40	GP Register
08.01.2022	10:00	IO Interface
11.01.2022	06:30	GP Bus
13.01.2022	07:00	Instruction Set & Decoder
20.01.2022	09:00	Decoder
23.01.2022	06:00	Program Memory
24.01.2022	05:00	Execution Cycle
28.01.2022	08:00	Assembler
29.01.2022	06:00	finishing Assembler & code highlighter
05.02.2022	12:00	Boot Loader & Boot Controller
06.02.2022	07:12	timings & finishing main
09.02.2022	04:12	timings & setting up latex
12.02.2022	06:00	learning latex, setting up documentation
18.02.2022	06:00	documentation
19.02.2022	10:30	documentation

The documented times account to around 205 hours. This however this does not include most of the time spent on documentation, due to not tracking the time anymore. The estimated total time is about 250 hours.