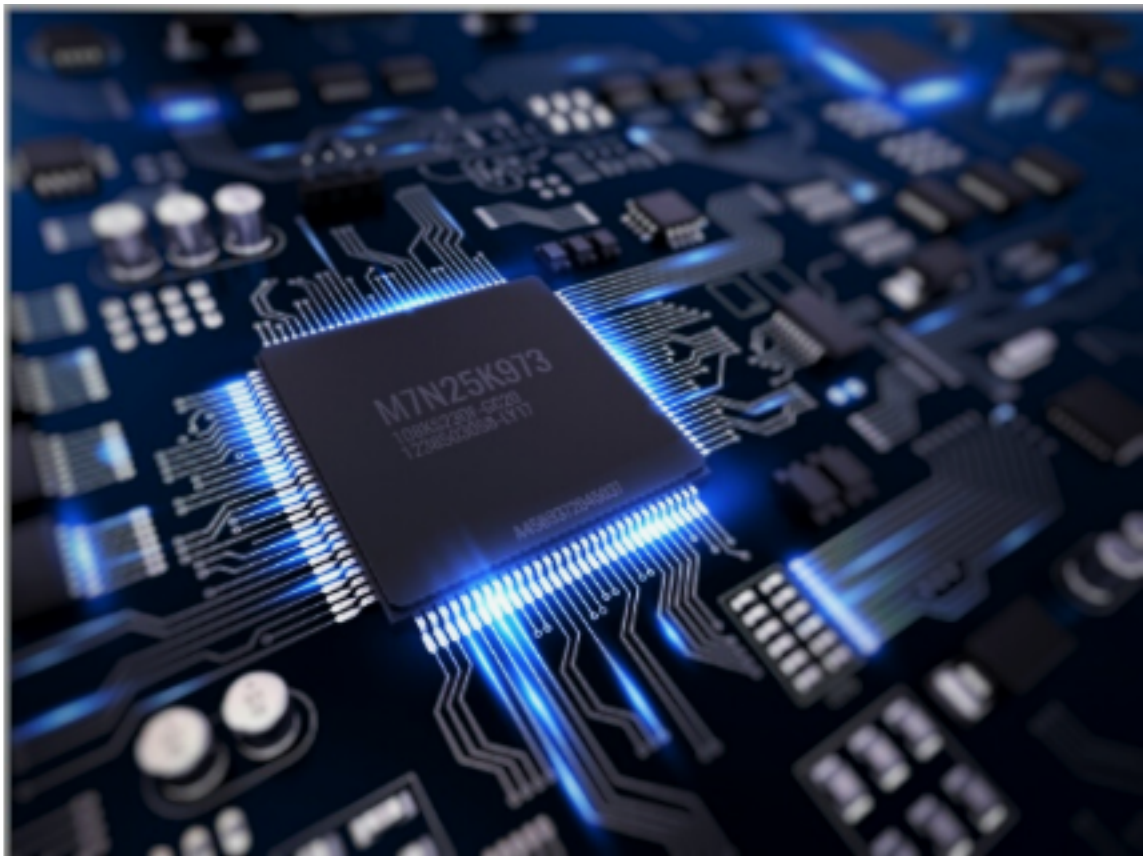
طراحی کامپیوتری سیستم های دیجیتال

پروژه پایانی
پیاده سازی میکروکنترلرهای AVR در VHDL

خاطره تاج فر - 9422353

بهار - 98



مقدمه

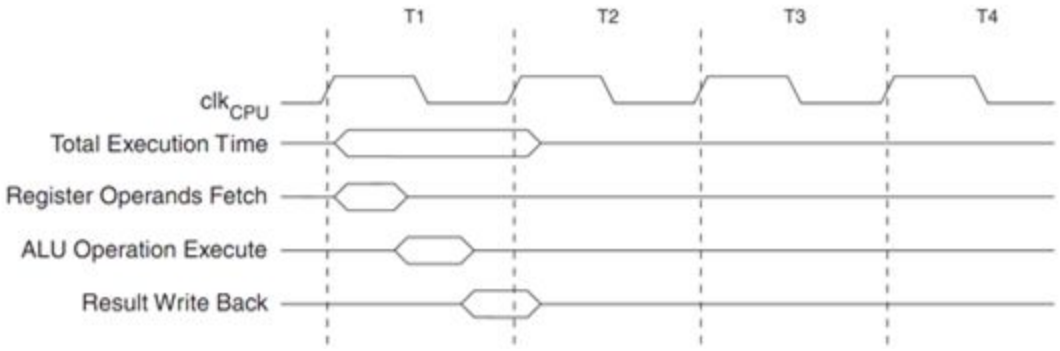
در این پروژه سعی شده است تا قسمت های کوچکی از این خانواده از میکروکنترلرها که کاربرد زیادی هم در صنعت دارند توسط زبان VHDL پیاده سازی شود که مشخصات آن به شرح زیر است :

- کاملا منطبق بر ATMega32 . دارای 32KByte Rom - 2Kbyte SRAM -32 Byte GPR

- پیاده سازی بیش از 20 دستورالعمل کلیدی (دستورات مشابه ، به راحتی قابل افزودن هستند)
- دارای Instruction Set کاملا منطبق بر AVR
- پیاده سازی Pipeline با عمق 2 دستورالعمل
- اجرای هر دستور در 1 کلاک سیکل
- دارای دستورات 2 خطی : Two Words Instructions
- قابلیت اجرای دستورات کامپایل شده در AVR Studio

1) ALU :

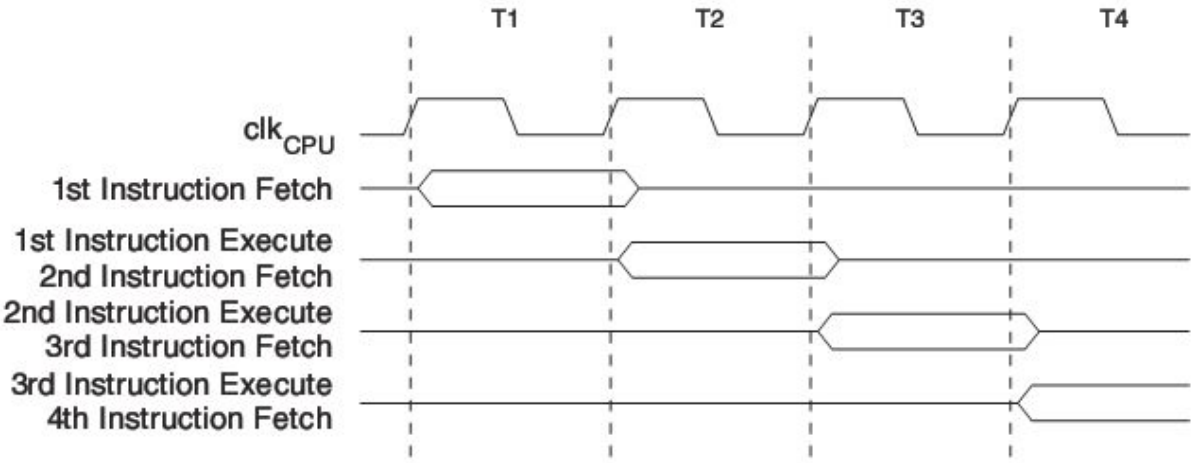
برای کاهش حجم سخت افزار مصرفی در این پروژه سعی شده است تا از یک بخش محاسباتی استفاده شود . این واحد با گرفتن دو ورودی 8 بیتی مستقیما از رجیستر ها و عملیات مورد نظر از طریق بیت های کنترلی آن ، حاصل را محاسبه کرده و در اختیار واحد کنترلی قرار میدهد . در ضمن فلگ های سیستم هم که در اختیار این واحد قرار میگیرند در محاسبات دخیل بوده و فلگ های جدید نیز برای ذخیره شدن در اختیار واحد کنترلی قرار میگیرند . این قسمت کاملا به صورت ترکیبی پیاده سازی شده است که افزایش سرعت را به همراه داشته است.



شکل 1 – نمودار زمانی محاسبه و ذخیره سازی واحد ALU

2) CU :

این واحد به صورت "رفتاری" پیاده سازی شده است که با اتصال SRAM ، PROM ، ALU و واحد های جانبی به آن کامل شده است . در این واحد دو پایپلاین به صورت دو ماشین حالت موازی عمل میکنند که خود باعث افزایش دو برابری سرعت پردازنده شده است .



شکل 2 – نمودار زمانی پایپ لاین Fetch- و Execution دو دستوره

دستورات پیاده سازی شده :

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1

EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
JMP	k	Jump	PC = k	None	3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic		Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register		Rd = Rr	None	1
LDI	Rd,K8	Load Immediate		Rd = K	None	1
LDS	Rd,k	Load Direct		Rd = (k)	None	2*
STS	k,Rr	Store Direct		(k) = Rr	None	2*
IN	Rd,P	In Port		Rd = P	None	1
OUT	P,Rr	Out Port		P = Rr	None	1

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
NOP	None	No operation	None	None	1

جدول 1 – دستورات پیاده سازی شده در پروژه

نحوه تست و شبیه سازی :

برای تست کافیست برنامه ای شامل دستورات فوق را در AVR Studio Assembler نوشته و پس از کامپایل کردن ، فایل هگز را در PROM طراحی شده وارد کنید.

کد وارد شده برای تست :

```
/*
 * AVRAssembler1.asm
 *
 *  Created: 7/28/2019 1:40:17 PM
 *   Author: KhatereTajfar
 */
 .org 0x0000
       jmp 0x0030
```

```
.org 0x0030

	ldi r16,10
	ldi r17,20
	ldi r20,10
	ldi r21,30
	add r17,r16
loop:
	cp r20,r16
	nop
	brbc 1,equal
	inc r16
	adc r17,r20
	jmp loop

.org 0x0045
	equal:
	dec r16
	mov r17,r20
	jmp loop
```

کد هگز تولید شده :

```
:020000020000FC
:040000000C9430002C
:100060000AE014E14AE05EE1100F4017000069F475
```

```
:080070000395141F0C943500E8
:08008A000A95142F0C943500B7
:00000001FF
```

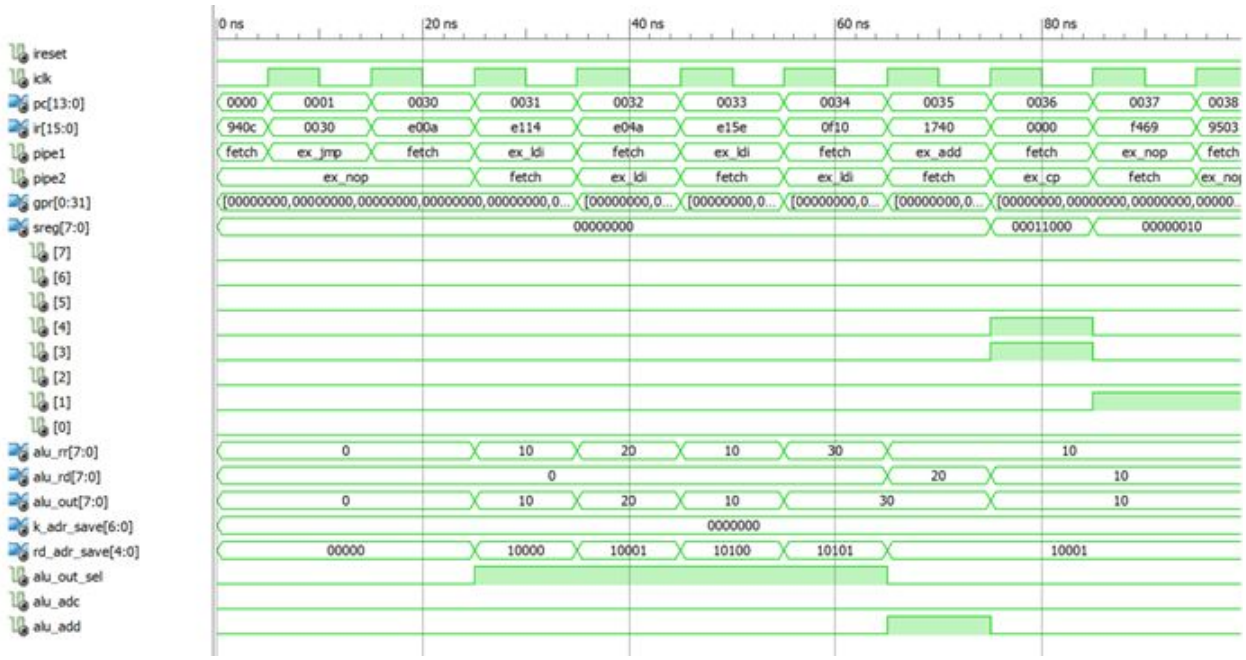
پس از استخراج کد ها :

```
x"940C" when address_in = 16#0000# else
x"0030" when address_in = 16#0001# else
x"E00A" when address_in = 16#0030# else
x"E114" when address_in = 16#0031# else
x"E04A" when address_in = 16#0032# else
x"E15E" when address_in = 16#0033# else
x"0F10" when address_in = 16#0034# else
x"1740" when address_in = 16#0035# else
x"0000" when address_in = 16#0036# else
x"F469" when address_in = 16#0037# else
x"9503" when address_in = 16#0038# else
x"1F14" when address_in = 16#0039# else
x"940C" when address_in = 16#003A# else
x"0035" when address_in = 16#003B# else
x"950A" when address_in = 16#0045# else
x"2F14" when address_in = 16#0046# else
x"940C" when address_in = 16#0047# else
x"0035" when address_in = 16#0048# else
x"ffff";
```

نتیجه شبیه سازی :

شکل ۶ – نتیجه شبیه سازی ALU

از شکل بالا نتیجه شد که واحد ALU به درستی کار میکند .



شکل 4– نتیجه شبیه سازی AVR Core

همانطور که از شکل فوق مشخص ایت دستورات مد نظر در دو پایپ لاین Pipe1 و Pipe2 پخش شده و اجرا می شوند . هر دستور از دو بخش : fetch و Execution تشکیل شده است و پایپ لاین به این صورت عمل میکند که وقتی Pipe1 در حال Fetch است Pipe 2 در حال Execution می باشد و بالعکس . برای دستورات پرش که نیاز به بیش از دو سیکل کاری دارند ، پایپلاین قالب که دستور پرش را در خود دارد پس از دیکد کردن OPCODE پایپلاین دیگر را در حالت غیر فعال یا NOP میبرد تا کار پرش به اتمام برسد .

دستورات دیگر نیز مطابق با روند بالا پس از آمدن در IR دیکد شده و برای اجرا آماده می شوند .

منابع مورد استفاده برای طراحی :

- ATMega 32 Full Datasheet
- ATMEL AVR Family Full Instruction Sets – Opcodes and address bits
- Internet