ЛПРС2 Лаб

Прозивка и Прекиди

верзија 2.0

Милош Суботић

15. март 2021.

1 Увод

Циљ вежбе је провежбати прозивку и прекиде над периферном јединицом, као и научити радити са стандардном периферном јединицом тајмера.

2 Тајмер

Тајмер TIMER има следеће главне регистре:

- `MAGIC` - кад се прочита даје `0xbabadeda`,
- `MODULO` - модуо по ком бројач броји. Бројач дакле броји у интервалу $[0, MODULO)$.
- `CNT` - тренутно стање тј. вредност бројача.
- `CTRL_STATUS` - регистар са пакованим заставицама (енгл. Flag).

Заставице осим што су паковане у `CTRL_STATUS` постоје и као засебни распаковани регистри. Заставице су следеће:

- `RESET` - Када је на 1, вредност бројача, односно регистар `CNT` се поставља на 0.
- `PAUSE` - Када је на 1, бројач не увећава своју вредност.
- `WRAP` - Индикатор да завршетка бројања (енгл. TC - Terminal Count) и да се бројач "премотава"на почетак. У том тренутку вредност бројача је $MODULO - 1$, а у наредном такту ће бити 0. У емулатору овај бит не ради.
- `WRAPPED` - Заставица која памти да је дошло до премотавања. Ако се десило премотавање заставица ће бити постављена на 1. Потребно је ручно је обрисати на 0.

Као што се може видети из `app_01_polling.c` апликације, тајмер има свој показивач преко којег се приступа регистрима:

```
12 #define timer_p32 ((volatile uint32_t*)TIMER_BASE)
```

као и своје макрое са индексима регистара и заставица за лакши приступ истим:

```
19 #define TIMER_CNT           0
20 #define TIMER_MODULO        1
21 #define TIMER_CTRL_STATUS   2
22 #define TIMER_MAGIC         3
23 #define TIMER_RESET_FLAG    0
24 #define TIMER_PAUSE_FLAG    1
25 #define TIMER_WRAP_FLAG     2
26 #define TIMER_WRAPPED_FLAG  3
27 #define TIMER_RESET         (TIMER_RESET_FLAG+4)
28 #define TIMER_PAUSE         (TIMER_PAUSE_FLAG+4)
29 #define TIMER_WRAP          (TIMER_WRAP_FLAG+4)
30 #define TIMER_WRAPPED       (TIMER_WRAPPED_FLAG+4)
```

На почетку програма је пример иштампавања свих регистара:

```
48     printf("TIMER_CNT          = 0x%08x\n", timer_p32[TIMER_CNT])
   ;
49     printf("TIMER_MODULO       = 0x%08x\n", timer_p32[
   TIMER_MODULO]);
50     printf("TIMER_CTRL_STATUS  = 0x%08x\n", timer_p32[
   TIMER_CTRL_STATUS]);
51     printf("TIMER_MAGIC        = 0x%08x\n", timer_p32[TIMER_MAGIC
   ]);
52     printf("TIMER_RESET        = 0x%08x\n", timer_p32[TIMER_RESET
   ]);
53     printf("TIMER_PAUSE        = 0x%08x\n", timer_p32[TIMER_PAUSE
   ]);
54     printf("TIMER_WRAP         = 0x%08x\n", timer_p32[TIMER_WRAP]
   );
55     printf("TIMER_WRAPPED      = 0x%08x\n", timer_p32[
   TIMER_WRAPPED]);
```

Бројач ради на 12 MHz, што значи да је потребно да бројач изброји до 12000000 за једну секунду, као што је и постављено у овој линији:

```
62     timer_p32[TIMER_MODULO] = 12000000; // modulo.
```

Уписом 0 у контролни тј. статусни регистар бришу се ресет и паузна заставица.

```
63     timer_p32[TIMER_CTRL_STATUS] = 0; // Start it.
```

Погледати регистарске мапе у Поглављу 8 за више детаља.

3 7-сегментни дисплеји

Ради лепшег прегледа у задатак су додани седмо-сегменти дисплеји. Његов показивач је:

```
11 #define digits_p32 ((volatile uint32_t*)DIGITS_BASE)
```

а помагајући макро за 4 расута регистра и 1 паковани су:

```
#define SEGM_0              0
#define SEGM_1              1
#define SEGM_2              2
#define SEGM_3              3
#define SEGM_PACK           4
```

Пример писања доња 4 бита променљиве `cnt` на 0. сегмент и горња 4 бита на 1. сегмент је дан на излисту испод:

```
        digits_p32[SEGM_0] = cnt & 0xf;
        digits_p32[SEGM_1] = cnt>>4;
```

Верзија са пакованим приступом је испод:

```
        digits_p32[SEGM_PACK] = ((cnt>>4)<<8) | (cnt & 0xf);
```

Погледати регистарске мапе у Поглављу 8 за више детаља.

4 Прекидачи и ледаче

Треба напоменути да је регистарска мапа PIO периферије малко друкчијег распореда него у претходним вежбама. Превасходно нема двоструке функционалности на истим локацијама (читају се прекидачи, пишу ледаче). Дато је битско поље, да се олкашало њено разумевање:

```
typedef struct {
    // Unpacked.
    // reg 0-7
    uint32_t led_unpacked[8];
    // reg 8-15
    uint32_t sw_unpacked[8];
    // Packed.
    // reg 16
    unsigned led_packed : 8;
    unsigned sw_packed  : 8;
    unsigned            : 16;
    uint32_t babadeda[15];
} bf_pio;
```

Следећи макро увелико може олакшати приступ битским пољима:

```
#define pio (*((volatile bf_pio*)PIO_BASE))
```

тако да није потребно користити стрелицу:

```
    pio.led_packed = 0x81; // For debugging purposes.
```

Погледати регистарске мапе у Поглављу 8 за више детаља.

5 Прозивка

Апликација описана кодом `app_01_polling.c` даје један једноставан пример тајмера коришћењем прозивке (енгл. Polling). Након постављања модула и поштања бројача у рад, могуће га је користити за синхронизацију алгоритам игрице (читање контрола, рачунање стања, писање излаза), да се изврши по тачно одређеном интервалу (1 секунда у овом

примеру). У бесконачној петљи која представља пролаз кроз потребно је на почетку сваке итерације сачекати премотавање бројача. На тај начин се у итерацију алгоритма са 3 горенаведена корака улази тек након што је прошла бројач избројао до модула, у овом примеру кад је избројао 1 секунду. Зато је потребно вршити прозивку заставице `WRAPPED`, да ли је постављена на 1. Након тога потребно је обрисати исту на 0. Пример за коришћење одпакованих регистара за приступ горепоменутој заставици је на излисту испод:

```
71          // Over unpacked.
72          // Poll wrapped flag.
73          WAIT_UNITL_TRUE(timer_p32[TIMER_WRAPPED]);
74          // Clear wrapped flag.
75          timer_p32[TIMER_WRAPPED] = 0;
```

А ево и примера за коришћење паковани регистара:

```
77          // Over packed.
78          // Poll wrapped flag.
79          WAIT_UNITL_TRUE(timer_p32[TIMER_CTRL_STATUS] & 1<<
    TIMER_WRAPPED_FLAG);
80          // Clear wrapped flag.
81          timer_p32[TIMER_CTRL_STATUS] &= ~(1<<TIMER_WRAPPED_FLAG)
    ;
```

У овом примеру, у кораку рачунања стања, увећава се `cnt`, који представља "софтверски"бројач секунди. Исти се даље прикајује на два 7-сегмента дисплеја.

6 Прекиди

Апликација описана кодом `app_02_interrupt.c` даје напреднији пример тајмера са коришћењем прекида (енгл. IRQ - Interrupt). Да би се користири прекид, потребно је регистровати прекидну рутину тј. функцију која ће бити позвана када дође до прекида. У случају тајмера то ће се десити током премотавања, односно у овој апликацији када тајмер избројо једну секунду. Пример регистрације је дан на излисту испод:

```
80      // Init IRQ.
81      alt_ic_isr_register(
82          TIMER_IRQ_INTERRUPT_CONTROLLER_ID, //alt_u32 ic_id
83          TIMER_IRQ, //alt_u32 irq
84          timer_isr, //alt_isr_func isr
85          NULL, //void *isr_context
86          NULL //void *flags
87      );
```

Потребно је определити индетификатор контролера прекида (овде `TIMER_IRQ_INTERRUPT_CONTROLLER_ID`), индетификатор прекида (овде `TIMER_IRQ`), прекидну рутину (овде `timer_isr`). Могуће је проследити контекст прекида, као и неке заставице за фина подешавања рада контролера прекида. Контекст прекида може да послужи да се иста прекидна рутина користи за више различитих уређаја; практично представља неки вид објектно-орјентисаног програмирања.

Прекидна рутина је дана на:

```
49 static void timer_isr(void* context) {
50     static uint8_t cnt = 0;
51 
52     //////////////////
53     // Read inputs.
54 
55     //////////////////
56     // Calculate state.
57 
58     cnt++;
59 
60     //////////////////
61     // Write outputs.
62 
63     pio.led_packed = cnt;
64     digits_p32[SEGM_0] = cnt & 0xf;
65     digits_p32[SEGM_1] = cnt>>4;
66 }
```

Једини параметар прекидне рутине је горeописани контекст. Унутар прекидне рутине се извршавају иста 3 корака горепоменутог алгоритма са бројачем секунди и излазом на 2 дисплеја.

Са коришћењем прекида `main` функција остаје слободна да обавља неки други посао. Овде тако на пример исписује пар вредности бројача:

```
95     printf("timer_p32 cnt reg:\n");
96     for(int j = 0; j < 300; j++){
97         printf("%9d\n", (int)timer_p32[TIMER_CNT]);
98 
99         // Busy wait.
100        for(int i = 0; i < 10*1000*1000; i++){}
101    }
```

Може се видети како се бројач увећава и кад се приближи 12000000 почиње бројање од 0. На крају је ипак потребно поставити бесконачну петљу како се програм не би завршио и емулатор изашао:

```
104    while(1){}
```

Обратити пажњу да се прекидна рутина извршава у засебној нити, тако да ако се неке променљиве деле између прекидне рутине и `main` функције, потребно је да су `volatile`.

7 Задаци

Користити TIMER као извор догађаја за софтверско бројање. Табела 1 је легенда шта значе одређене скраћенице у спецификацији.

Табела 1: Легенда спецификације

32	преко 32-битног показивача (индексирање, макрои, маскирање)
bf	преко битског поља
p	користити паковане регистре
u	користити распаковане регистре

Колона у табели спецификације се избира на основу задња 3 бита индекса $idx[2:0]$. За DIGITS и TIMER постоје 4 групе регистарских мапа одређена са $idx[2:1]$.

7.1 *app_03_stopwatch*

Израдити штоперицу. Користити низ од 4 бајта за софтверско бројање цифара штоперице. Табела 2 дефинише која периферија има коју функцију: За излазне шта треба приказати на њима, а за улазне на шта утичу. Табела 3 дефинише који термин треба да користи коју јединицу на који начин.

Табела 2: Функције периферија *app_03_stopwatch*

Периферија	Регистар	Функција
7-сегменти дисплеји	`SEGM0`	Јединице стотинке
	`SEGM1`	Десетице стотинке
	`SEGM2`	Јединице секунде
	`SEGM3`	Десетице секунде
Ледаче	`LED[3:0]`	Десетице стотинке
	`LED[7:4]`	Јединице секунде
Прекидачи	`SW0`	Ресет
	`SW1`	Пауза

Табела 3: Спецификација за *app_03_stopwatch*

$idx[2:0]$	000	001	010	011	100	101	110	111
SW	bf p	32 u	bf u	32 p	bf p	32 u	bf u	32 p
LED	32 u	bf p	32 p	bf u	32 u	bf p	32 p	bf u
7SEGM	bf p	32 u	bf p	32 u	bf u	32 p	bf u	32 p
TIMER	32 u	bf p	32 u	bf p	32 p	bf u	32 p	bf u
IRQ vs Poll	irq	poll	irq	poll	irq	poll	irq	poll

7.2 *app_04_chess_clock*

Израдити сат за шах. Користити 2 низа од по 2 бајта за софтверско бројање цифара. Табела 4 дефинише која периферија има коју функцију: За излазне шта треба приказати на њима, а за улазне на шта утичу. Табела 5 дефинише који термин треба да користи коју јединицу на који начин.

Табела 4: Функције периферија *app_04_chess_clock*

Периферија	Регистар	Функција
7-сегменти дисплеји	`SEGM0`	Јединице секунде белог играча
	`SEGM1`	Десетице секунде белог играча
	`SEGM2`	Јединице секунде црног играча
	`SEGM3`	Десетице секунде црног играча
Ледаче	`LED0`	Активан бели играч
	`LED7`	Активан црни играч
Прекидачи	`SW0`	Ресет
	`SW1`	Пауза
	`SW7`	Активан: 0 - бели, 1 - црни

Табела 5: Спецификација за *app_04_chess_clock*

$idx[2:0]$	000	001	010	011	100	101	110	111
SW	32 u	bf p	32 p	bf u	32 u	bf p	32 p	bf u
LED	bf p	32 u	bf u	32 p	bf p	32 u	bf u	32 p
7SEGM	32 u	bf p	32 u	bf p	32 p	bf u	32 p	bf u
TIMER	bf p	32 u	bf p	32 u	bf u	32 p	bf u	32 p
IRQ vs Poll	poll	irq	poll	irq	poll	irq	poll	irq

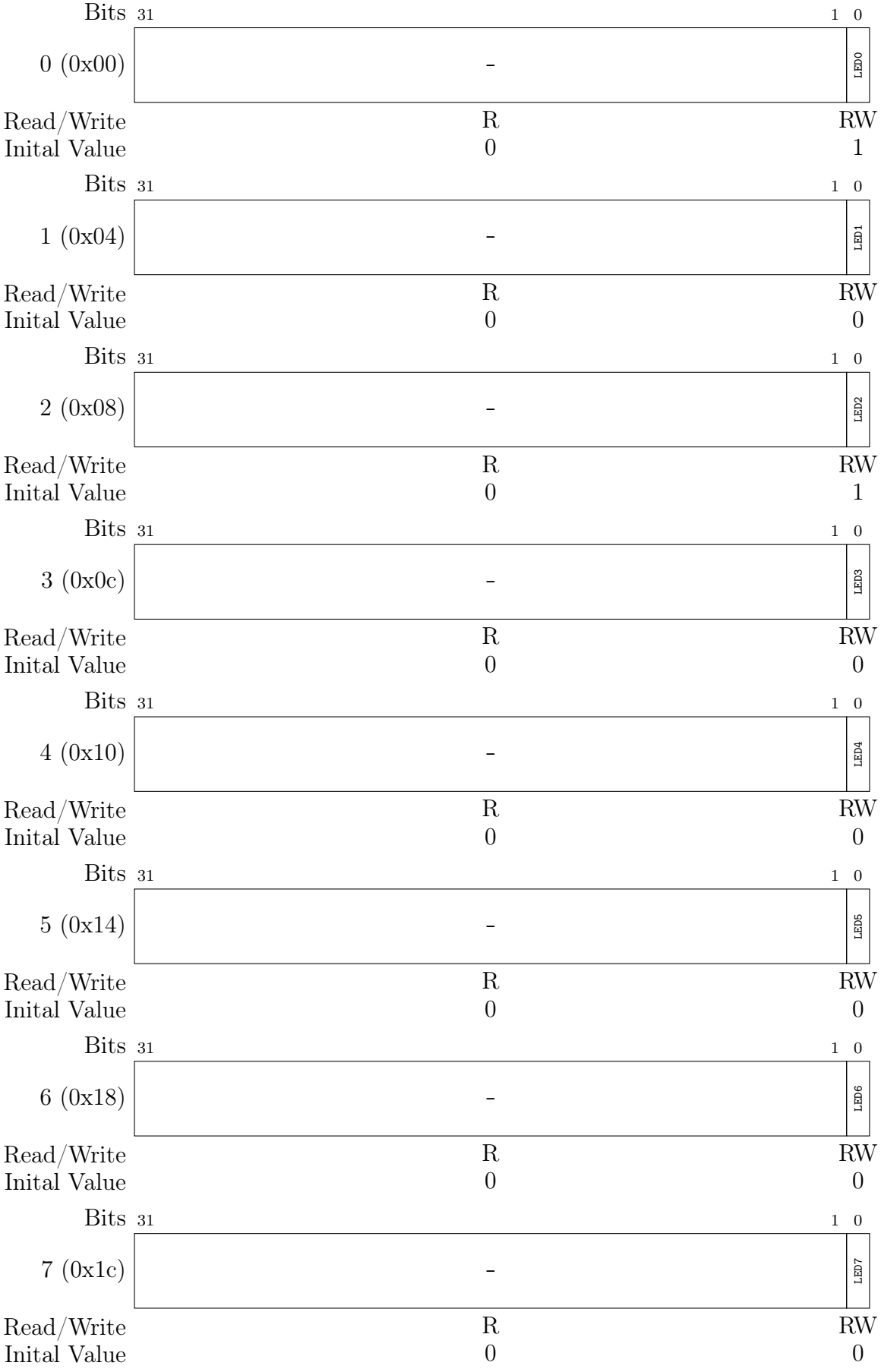
8 Регистарске мапе

Регистарска мапа за `PIO` (прекидачи и ледаче) је иста за све термине.

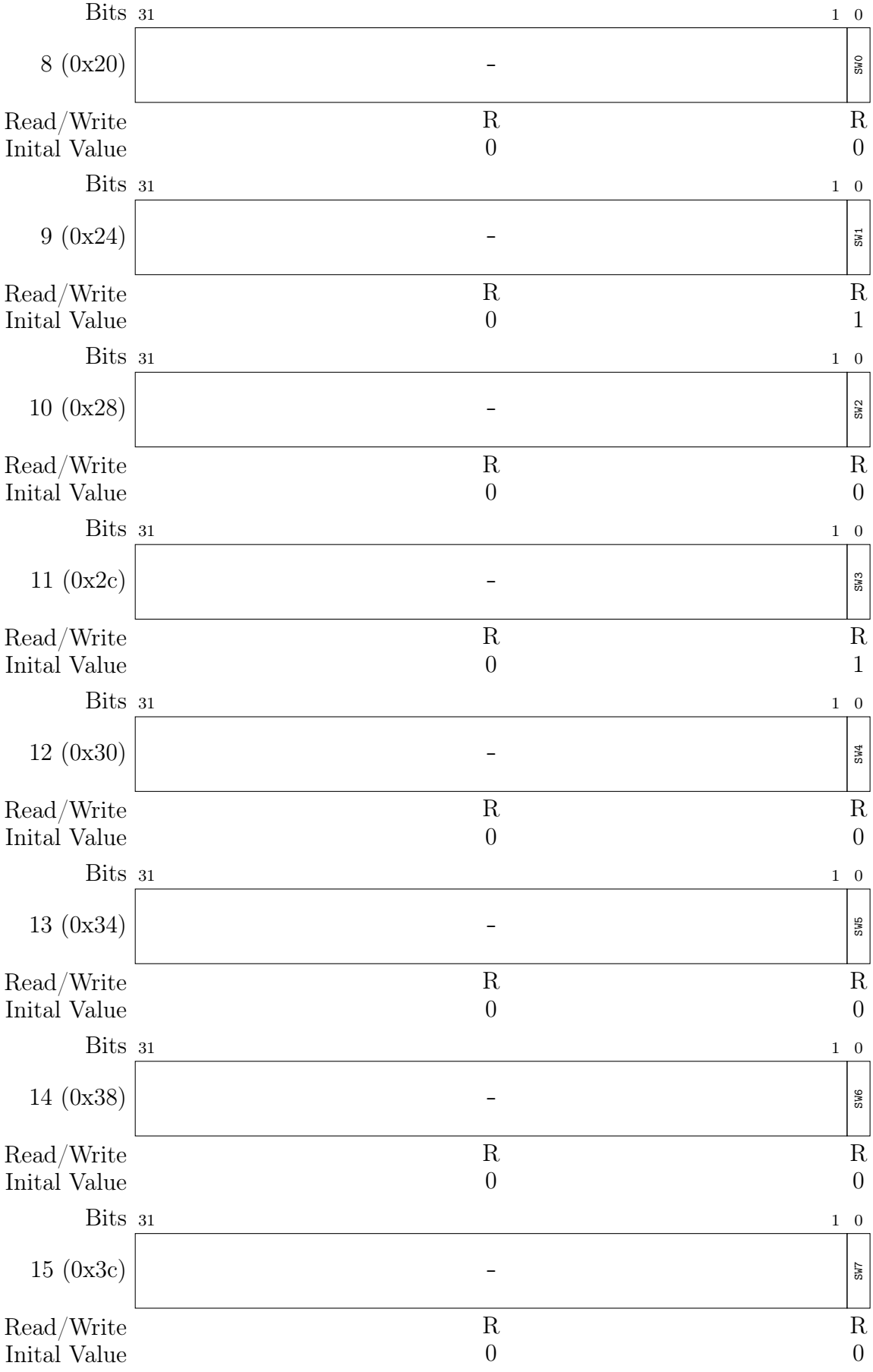
За тајмер (TIMER) и 7-сегменти дисплеј DIGIT постоје 4 групе одређене са $idx[2:1]$ и свака група има своју меморијску мапу за ове две компоненте. Притом, нулти регистар почиње на различитим адресама. асдф **Нпр.** тајмер за групу $idx[2:1]$=0b00 почиње на `TIMER_BASE` док за **за групу** $idx[2:1]$=**0b01 на** `TIMER_BASE+0x20`. Такође, **редослед регистара није исти**. У `app_02_interrupt.c` је пример како се постављају вредности за одговарајуће показиваче:

```
12 #define digits_p32 ((volatile uint32_t*)(DIGITS_BASE+0x20))
13 #define timer_p32 ((volatile uint32_t*)(TIMER_BASE+0x20))
```

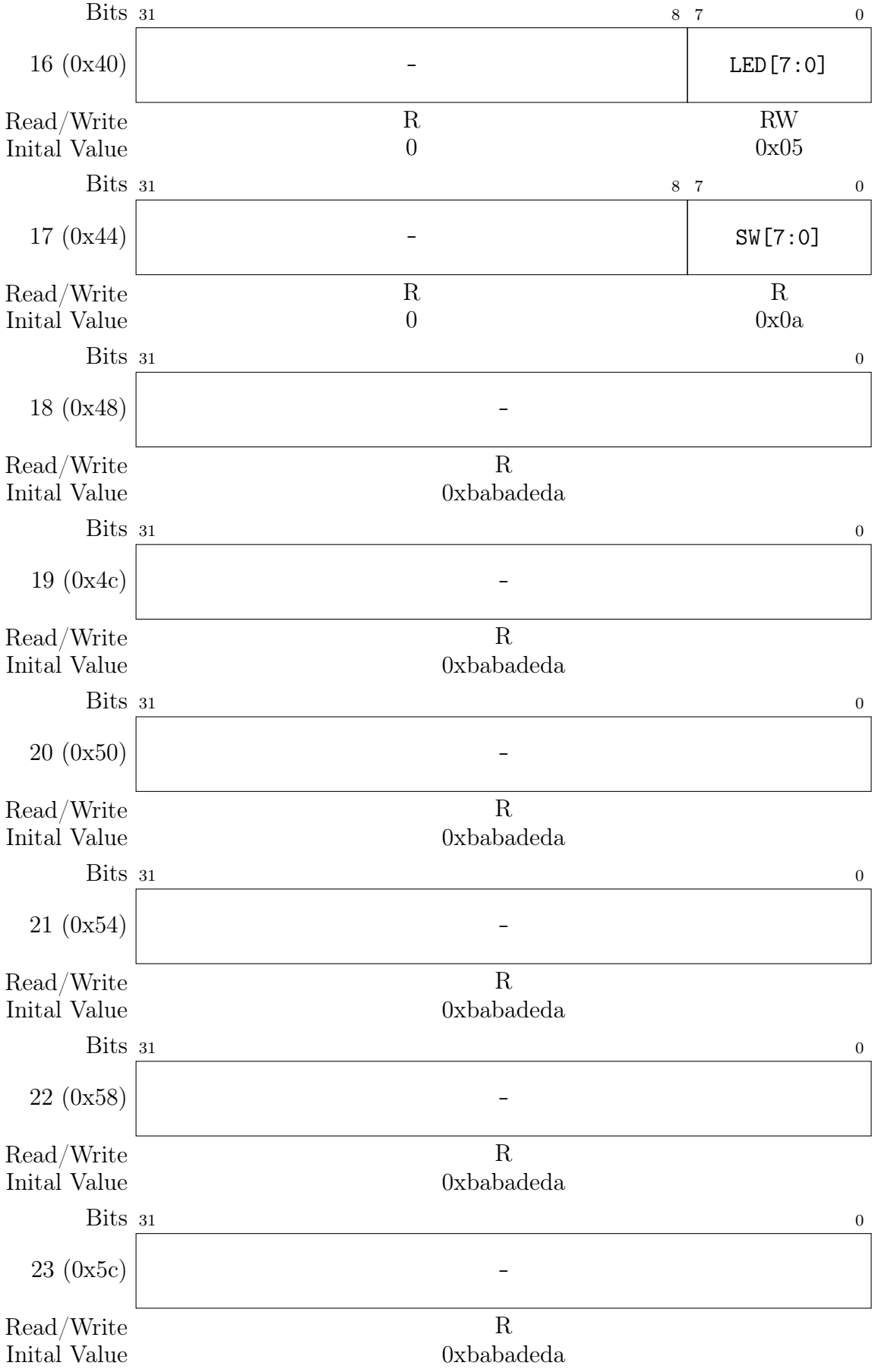
. Приметити такође да су помагајући макрои коришћени у апликацији `app_02_interrupt.c` са друкчијим вредностима у односу на `app_01_polling.c`, услед другчијег размештаја регистара у регистарској мапи.



Слика 1: PIO меморијска мапа 1/4



Слика 2: PIO меморијска мапа 2/4



Слика 3: PIO меморијска мапа 3/4

Bits	31	0
24 (0x64)	-	
Read/Write	R	
Inital Value	0xbabadeda	

Bits	31	0
25 (0x64)	-	
Read/Write	R	
Inital Value	0xbabadeda	

Bits	31	0
26 (0x68)	-	
Read/Write	R	
Inital Value	0xbabadeda	

Bits	31	0
27 (0x6c)	-	
Read/Write	R	
Inital Value	0xbabadeda	

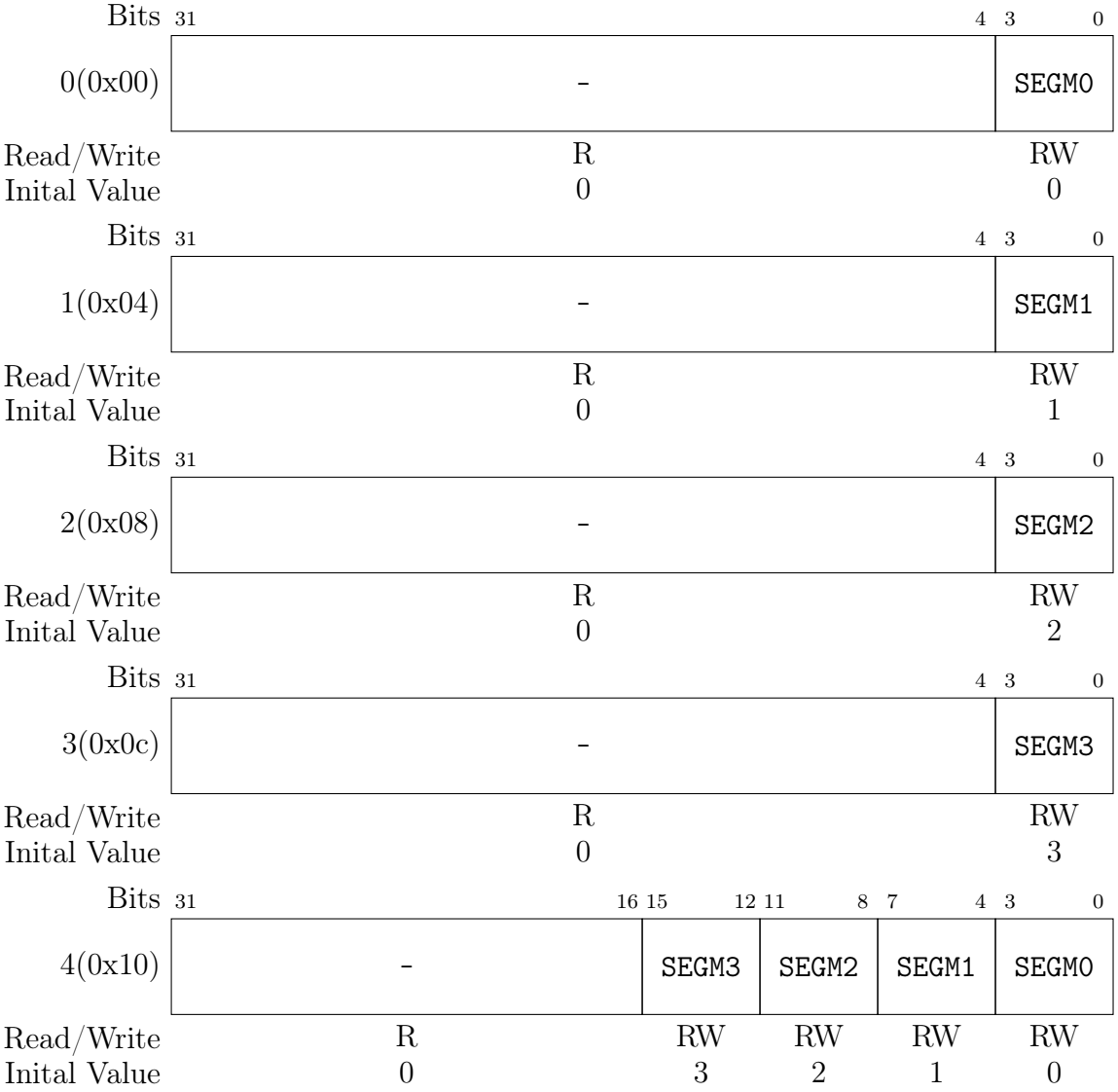
Bits	31	0
28 (0x70)	-	
Read/Write	R	
Inital Value	0xbabadeda	

Bits	31	0
29 (0x74)	-	
Read/Write	R	
Inital Value	0xbabadeda	

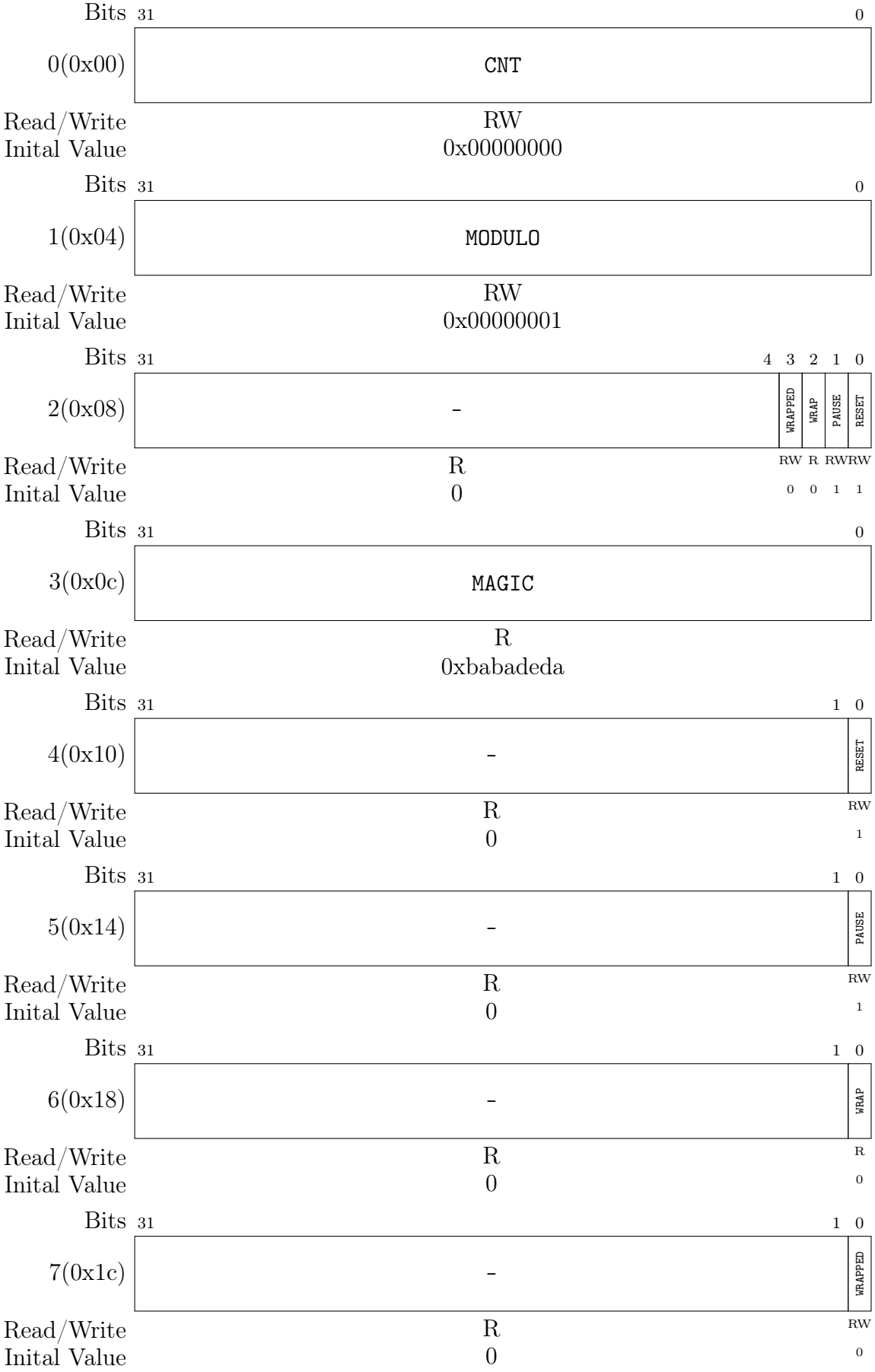
Bits	31	0
30 (0x78)	-	
Read/Write	R	
Inital Value	0xbabadeda	

Bits	31	0
31 (0x7c)	-	
Read/Write	R	
Inital Value	0xbabadeda	

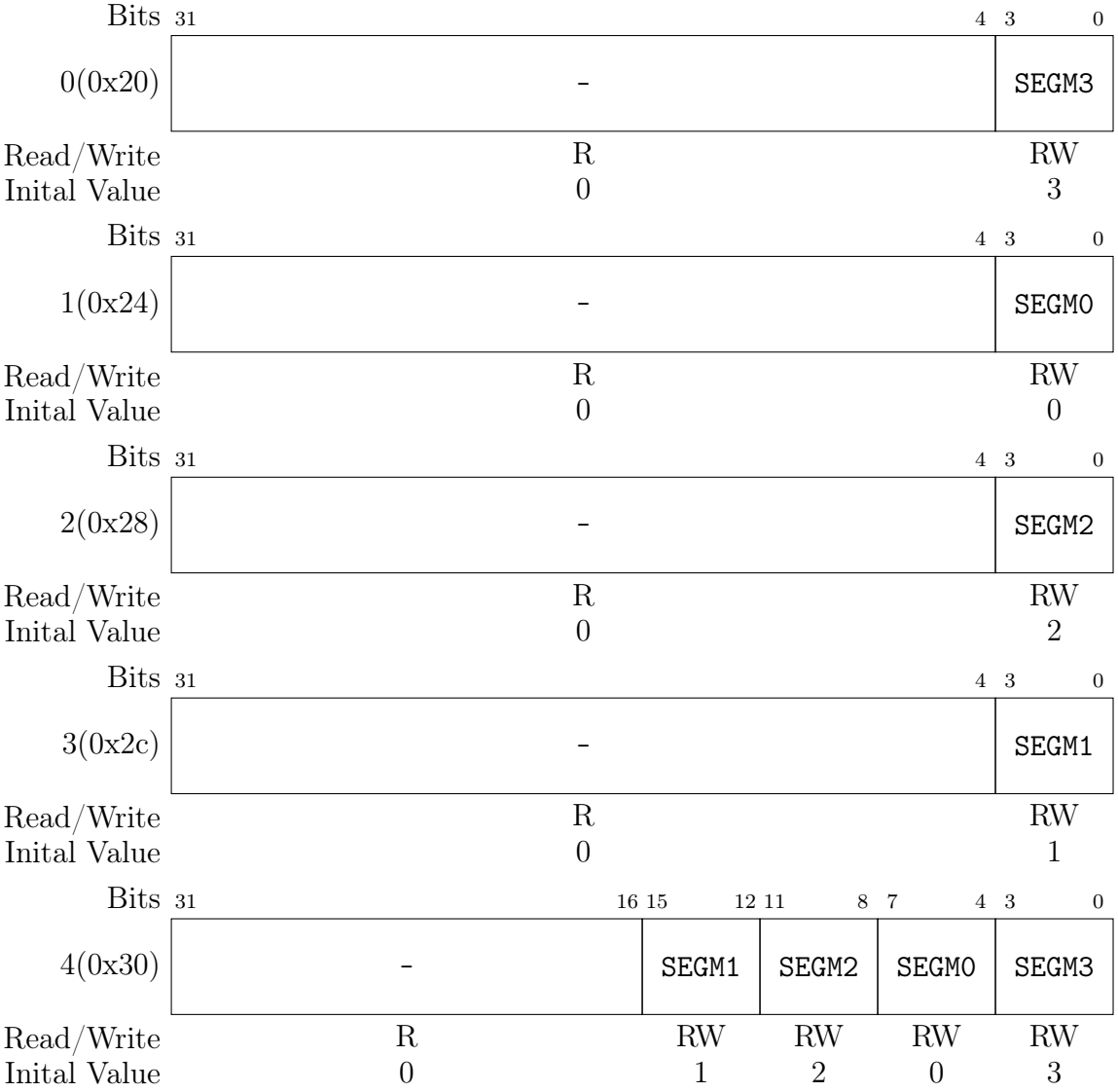
Слика 4: PIO меморијска мапа 4/4



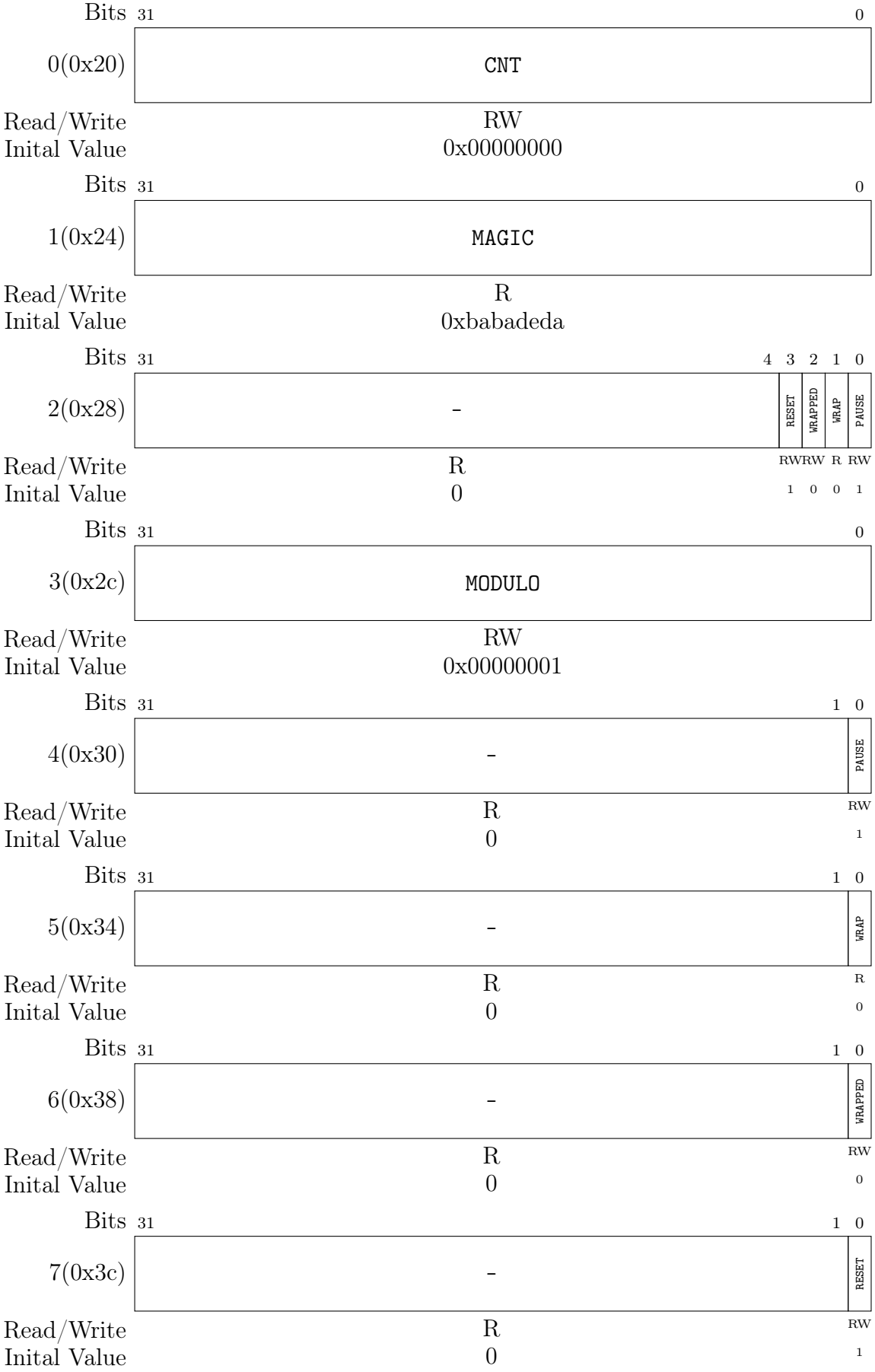
Слика 5: DIGITS меморијска мапа за групу $idx[2:1]$=0b00



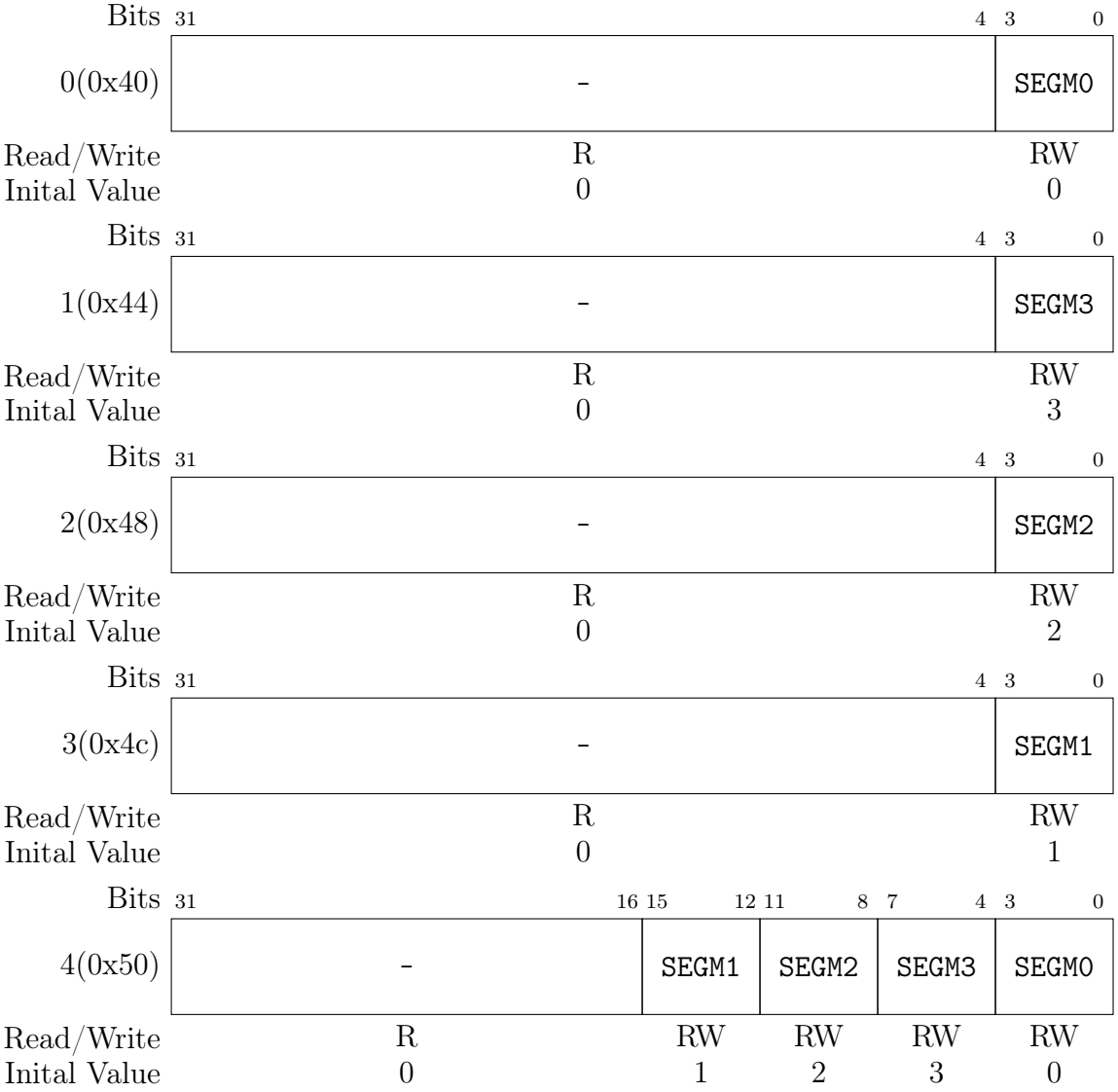
Слика 6: TIMER меморијска мапа за групу $idx[2:1]$=0b00



Слика 7: DIGITS меморијска мапа за групу $idx[2:1]$=0b01

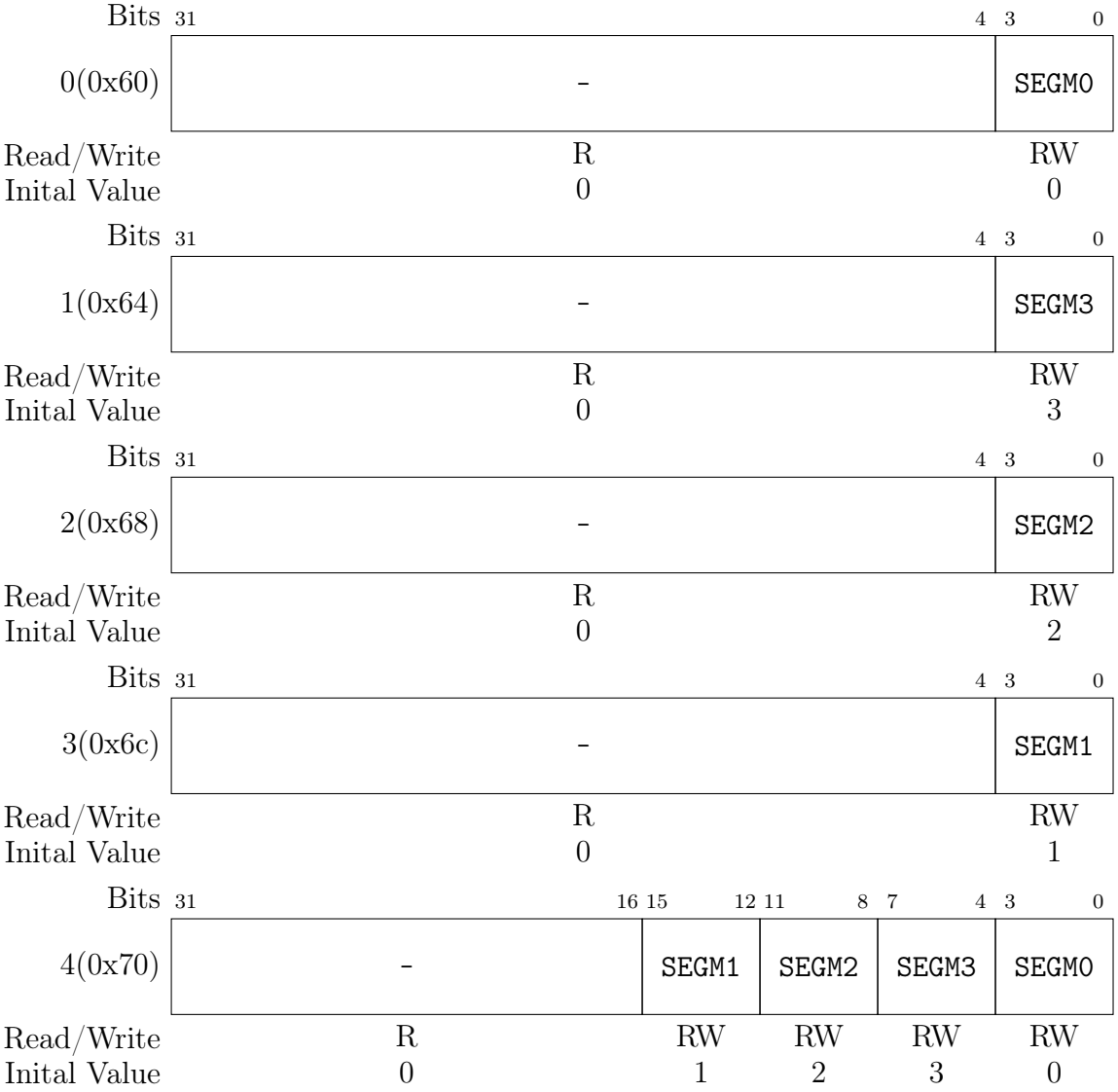


Слика 8: TIMER меморијска мапа за групу $idx[2:1]$=0b01

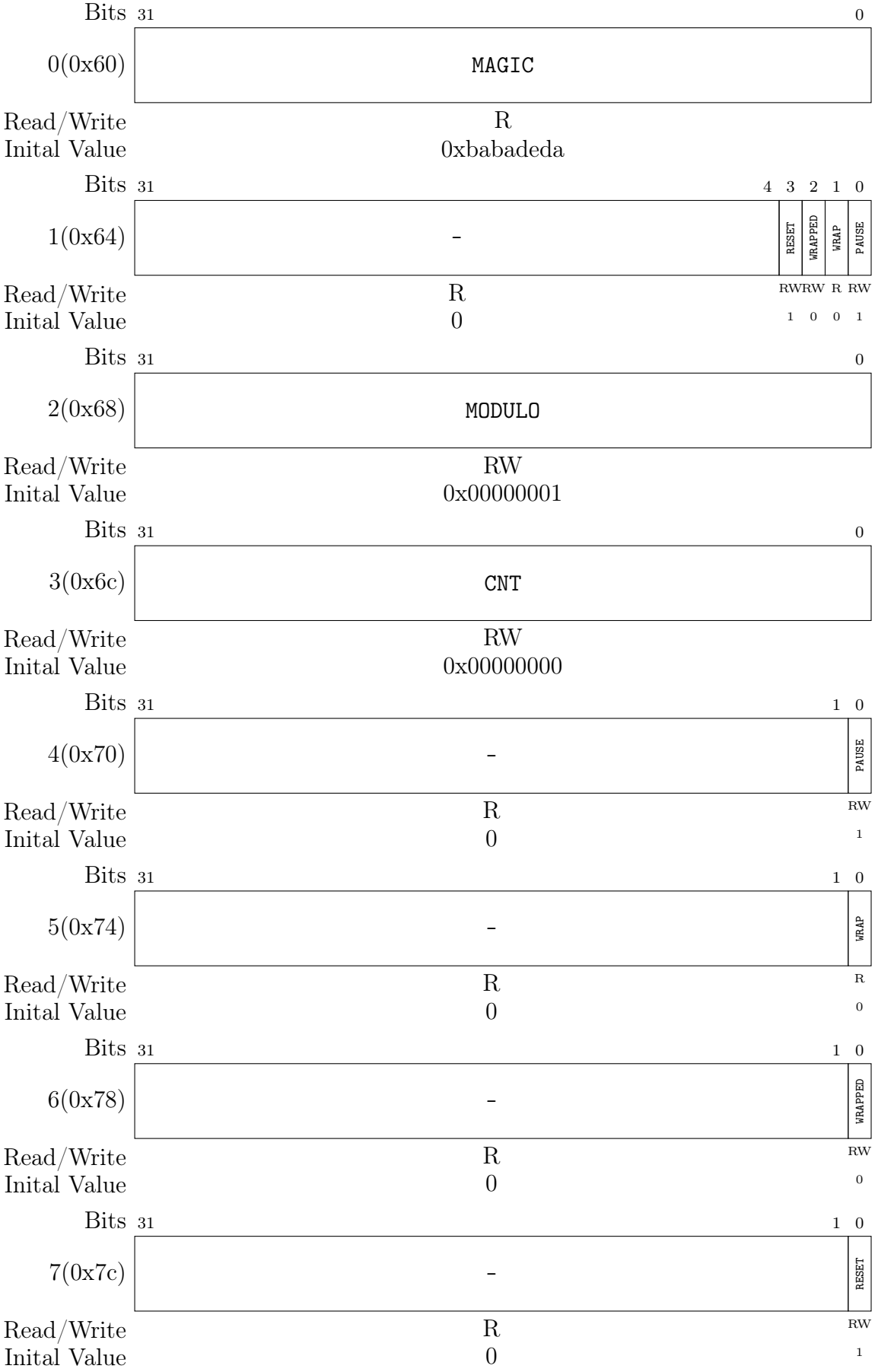


Слика 9: DIGITS меморијска мапа за групу $idx[2:1]$=0b10

Слика 10: TIMER меморијска мапа за групу $idx[2:1]$=0b10



Слика 11: DIGITS меморијска мапа за групу $idx[2:1]$=0b11



Слика 12: TIMER меморијска мапа за групу $idx[2:1]$=0b11