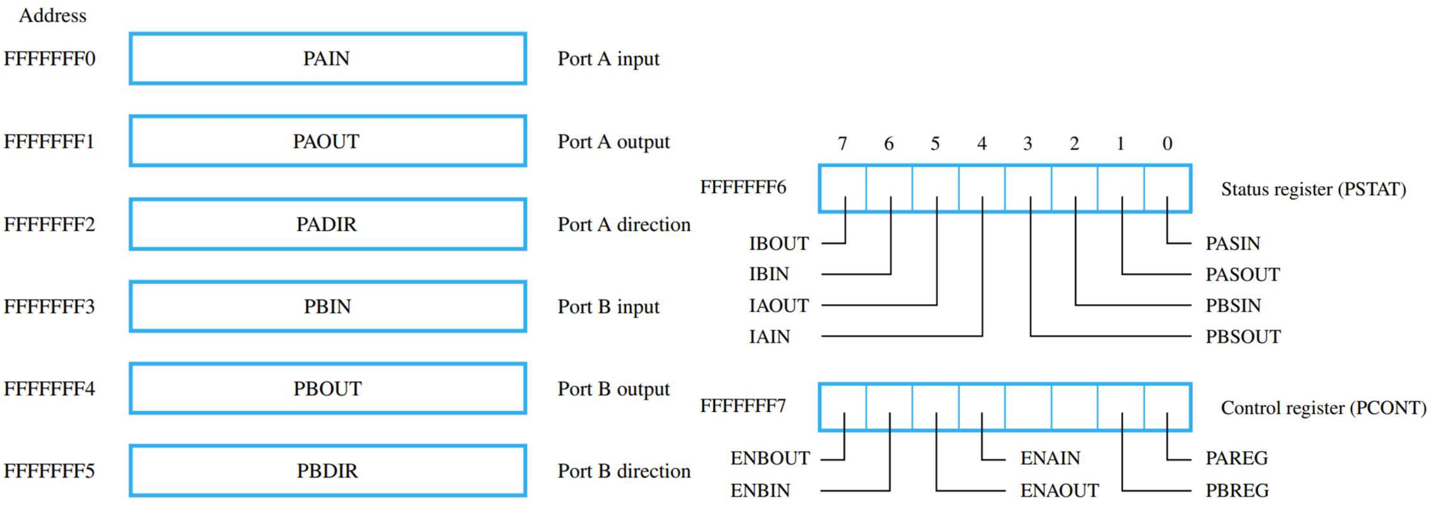
Textbook's Microprocessor:

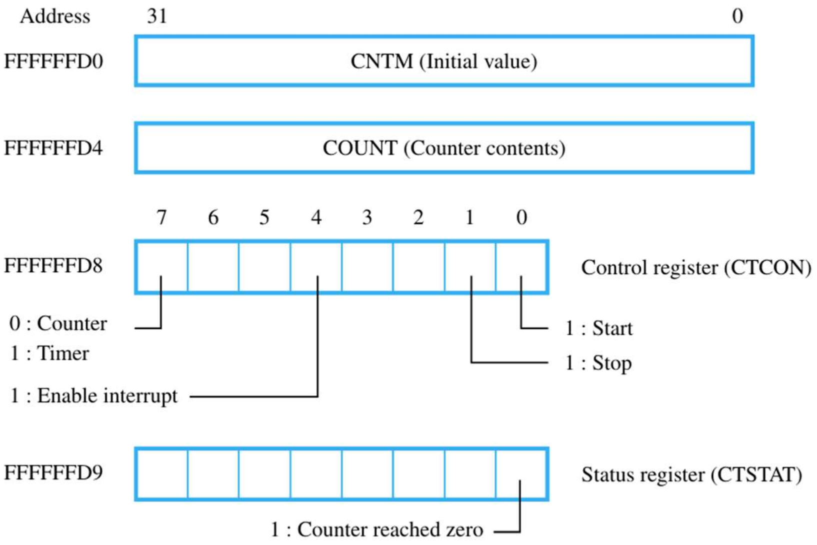


For PADIR, PBDIR: 0 -> input, 1 -> output

For PASIN/PBSIN: 1 -> new data are in PAIN or PBIN, 0 -> new data has been read

For PASOUT/PBSOUT: 1 -> data from PAOUT or PBOUT have been accepted by external device, 0 -> new data has been written

PCONT -> used to enable interrupts on change of P(A/B)IN or P(A/B)OUT, PCONT[0]-[3] should be set to 0



Textbook counter always counts down.

Use CNTM to change length of timer. Textbook timer runs at 100MHz, so set CNTM = 100,000,000 (w/o commas) to set length to 1 second.

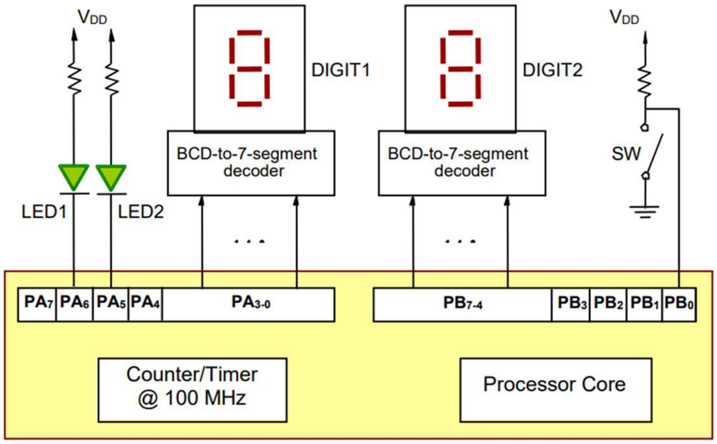
CTCON [7] = 0 (use counter mode, not timer mode)

Setting CTCON[0] = 0 does not stop the timer, must set CTCON[1] = 1

If polling, check for counter reached 0 flag = 1 to tell when time has elapsed

Must reset counter flag manually

Coding example: Timer interrupt, I/O main loop



Task 1 (main loop):

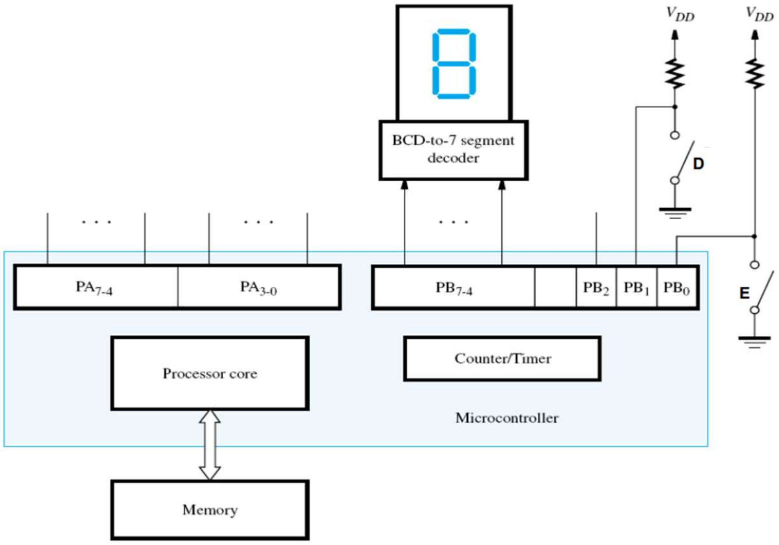
- Whenever button is pressed and released, LED 1 and 2 should swap states

Task 2 (interrupt):

- Every second, decrement DIGIT1 if LED 1 is on, or decrement DIGIT2 if LED2 is on. (decrementing 9 gives 0)

```
interrupt void intserv();

volatile unsigned char display_1 = 0;
volatile unsigned char display_2 = 0;

volatile unsigned char led_1_on = 1;

int main() {
  *CTCON = 0x2; /* stop timer */
  *CTSTAT = 0x0; /* clear reached 0 flag */

  *PADIR = 0xFF; /* all out*/
  *PBDIR = 0xF0; /* ports [7]-[4] out, [3]-[0] in (only care about [0])*/

  *PAOUT = 0xB0; /* set display = 0, LED1 = on and LED2 = off (1011 0000)*/
  *PBOUT = 0x0; /* set display = 0*/

  *CNTM = 100000000; // 100MHz counter -> 100,000,000 cycles per second

  *IVECT = (unsigned int *) &intserv; /* set contents of address IVECT to memory address of intserv function */
  asm("MoveControl PSR, #0x40"); /* sets PSR[6] = 1, enabling interrupts */

  *CTCON = 0x11; /* start timer, enable interrupts */

  while(1) {
    if ((*PBIN | 0x1) == 0) { // if button is pressed
      while((*PBIN | 0x1) == 0); // loop while button held

      // when button released
      if (led_1_on) {
        *PAOUT = (unsigned char)(0xD0 | (*PAOUT & 0x0F)); // keep PAOUT[0]-[3], set bits [4]-[7] = 1101
        led_1_on = 0;
      } else {
        *PAOUT = (unsigned char)(0xB0 | (*PAOUT & 0x0F)); // keep PAOUT[0]-[3], set bits [4]-[7] = 1011
        led_1_on = 1;
      }
    }
  }
  exit(0);
}

interrupt void intserv() {
  *CTSTAT = 0x0; /* reset reached 0 flag */
  if (led_1_on) {
    display_1 = display_1 == 0 ? 9 : display_1 - 1; // if subtracting 1 would make display roll over, set to 9 instead
    *PAOUT = (unsigned char)((*PAOUT & 0xF0) | display_1); // keep PAOUT[4]-[7], set bits [0]-[3] = display[0]-[3]
  } else {
    display_2 = display_2 == 0 ? 9 : display_2 - 1; // if subtracting 1 would make display roll over, set to 9 instead
    *PBOUT = (display_2 << 4); // 0000 XXXX -> XXXX 0000 (shift bits left by 4)
  }
}
```

Coding example: Timer loop, I/O interrupt



Task 1 (main loop):

- Increment digit every second if E was pressed last. If D was pressed last, don’t increment (incrementing 9 gives 0)

Task 2 (interrupt):

- Interrupt sent when PBIN updated. If the update was D being pressed, disable incrementing digit. If the update was E being pressed, enable incrementing digit

```
interrupt void intserv();

unsigned char display = 0;

int main() {
  *CTCON = 0x2; /* stop timer */
  *PBDIR = 0xF3; /* ports [7]-[4] out, [0] and [1] in*/
  *PBOUT = 0x0; /* set display = 0*/
  *PCONT = 0x40; /* enable interrupts for port B in (ENBIN = 1)*/
  *CNTM = 100000000; // 100MHz counter -> 100,000,000 cycles per second

  *IVECT = (unsigned int *) &intserv; /* set contents of address IVECT to memory address of intserv function */
  asm("MoveControl PSR, #0x40"); /* sets PSR[6] = 1, enabling interrupts */

  while(1) {
    while((*CTSTAT & 0x1) == 0x0); /* do nothing until 0 is reached*/

    *CTSTAT = 0x0; /* reset reached 0 flag */
    display = (display + 1) % 10;
    *PBOUT = (display << 4); // 0000 XXXX -> XXXX 0000 (shift bits left by 4)
  }

  exit(0);
}

interrupt void intserv() {
  if ((*PBIN & 0x02) == 0) *CTCON = 0x2; /* if disable button pressed (PBIN[1] == 0) stop timer */
  else if ((*PBIN & 0x01) == 0) *CTCON = 0x1; /* if enable button pressed (PBIN[0] == 0), start timer */
}
```

Scheduling:

Priority algorithms:

Rate monotonic	Deadline monotonic	Earliest deadline first	Least laxity first
$\tau_{\mathrm{ik}} = \frac{1}{P_{\mathrm{i}}}$	$\tau_{\mathrm{ik}} = \frac{1}{D_{\mathrm{i}}}$	$\tau_{\mathrm{ik}} = \frac{1}{\phi_i + kP_{\mathrm{i}} + D_i}$	$\tau_{\mathrm{ik}} = \frac{1}{\phi_i + kP_{\mathrm{i}} + D_i - t - \Delta C_i}$

Where ΔC_i is the remaining execution time of task i

CPU utilisation = $C_1/P_1 + C_2/P_2 + \ldots$

Steps for solving problems:

Step 1: Determine max time period to consider (LCM of task periods)

Step 2: Mark when tasks are added to the queue on the diagram

Step 3: Calculate priorities using priority algorithm

Step 4: Consider task priorities every time a new task is added (write out the current queue to determine next task to be scheduled if needed)

Step 5: Check to make sure all tasks finish before their deadlines

E.g. Show the task schedule using the Earliest Deadline First (EDF) priority assignment. Note: If some tasks happen to have the same EDF priority, break such ties using Rate Monotonic (RM) prioritization.

Task $\mathbf{T}_i$	Period $\mathbf{P}_i$	WCET $\mathbf{C}_i$	Deadline $\mathbf{D}_i$	Initial delay ϕ_i
T1	30	10	30	0
T2	40	10	40	0
T3	60	10	50	0
T4	120	15	100	0

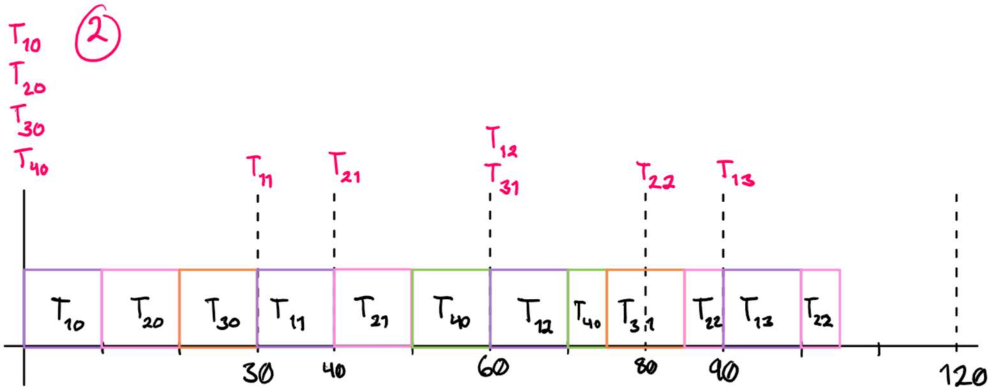
Step 1: LCM = 120

Step 2: See pink text in diagram

Step 3: See orange text in diagram

Step 4: See coloured squares in diagram

2



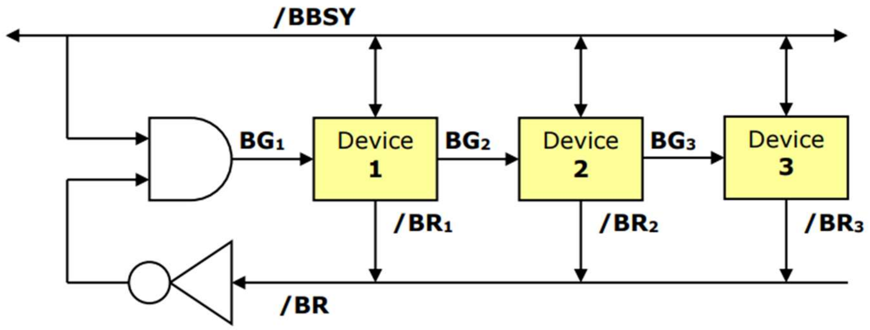
3 $\tau_{10} = \frac{1}{0 \cdot 30 + 30} = \frac{1}{30}$ $\tau_{11} = \frac{1}{1 \cdot 30 + 30} = \frac{1}{60}$ $\tau_{12} = \frac{1}{2 \cdot 30 + 30} = \frac{1}{90}$ $\tau_{13} = \frac{1}{3 \cdot 30 + 30} = \frac{1}{120}$

$\tau_{20} = \frac{1}{0 \cdot 40 + 40} = \frac{1}{40}$ $\tau_{21} = \frac{1}{1 \cdot 40 + 40} = \frac{1}{80}$ $\tau_{22} = \frac{1}{2 \cdot 40 + 40} = \frac{1}{120}$

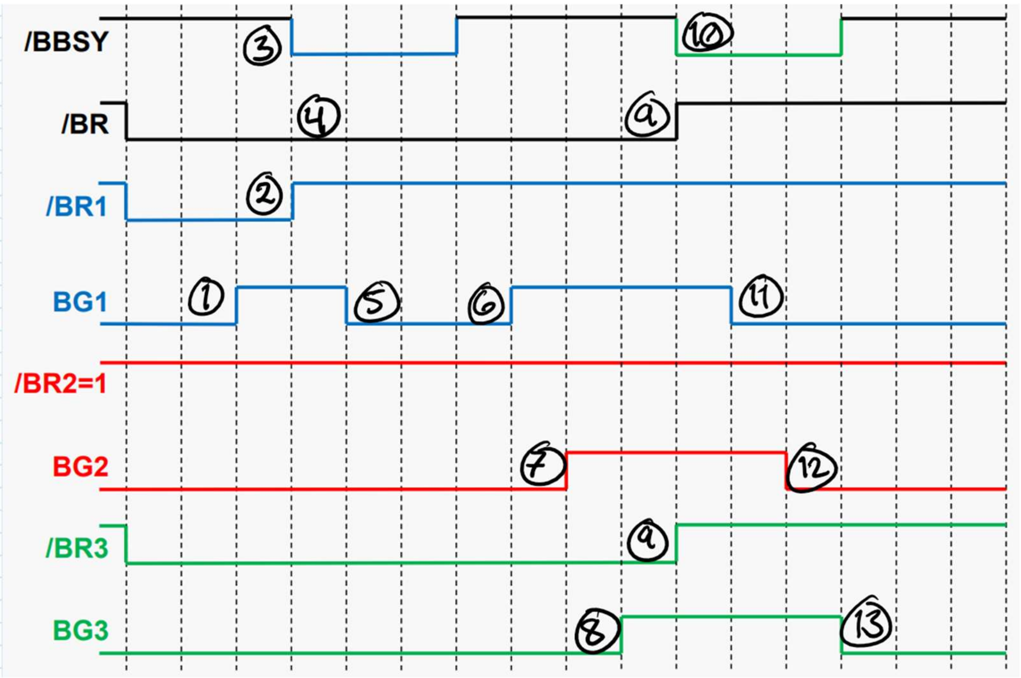
$\tau_{30} = \frac{1}{0 \cdot 60 + 50} = \frac{1}{50}$ $\tau_{31} = \frac{1}{1 \cdot 60 + 50} = \frac{1}{110}$

$\tau_{40} = \frac{1}{0 \cdot 120 + 100} = \frac{1}{100}$

Daisy chain arbitration timing diagrams:



Note: “/” means active/asserted low



1. AND gate sees /BR and /BBSY during first time interval and takes one time delay to assert BG1

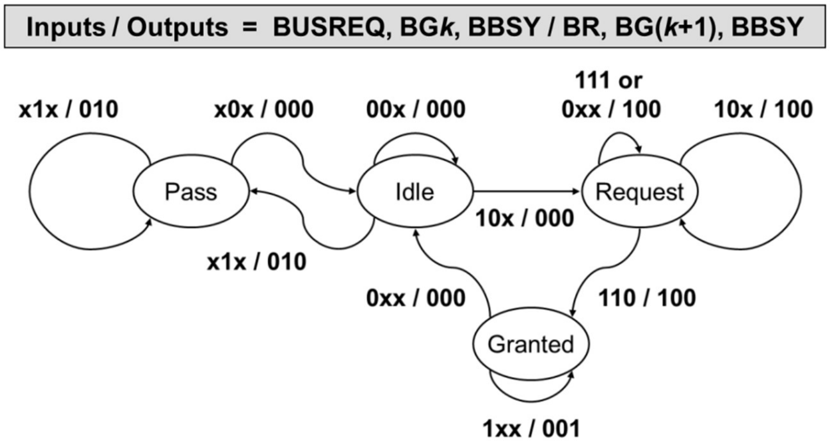
2. /BR1 takes one time delay after BG1 to stop asserting
3. /BBSY also takes one time delay after BG1 to assert
4. /BR remains asserted because /BR3 is still active
5. BG1 goes low one time delay after /BBSY stops asserting (one delay through AND gate)
6. Since there's still a request, BG1 goes high one time delay after /BBSY has stopped asserting
7. Device 1 doesn't want the signal, so BG2 goes high after one delay
8. Device 2 doesn't want the signal, so BG3 goes high after one delay
9. /BR3 takes one time delay after BG3 to stop asserting. Since /BR3 is directly connected to /BR and no other devices are requesting, /BR stops being asserted immediately.
10. /BBSY also takes one time delay after BG3 to assert
11. /BBSY is active, so after one delay BG1 stops being asserted
12. BG1 isn't active, so after one delay BG2 stops being asserted
13. BG2 isn't active, so after one delay BG3 stops being asserted

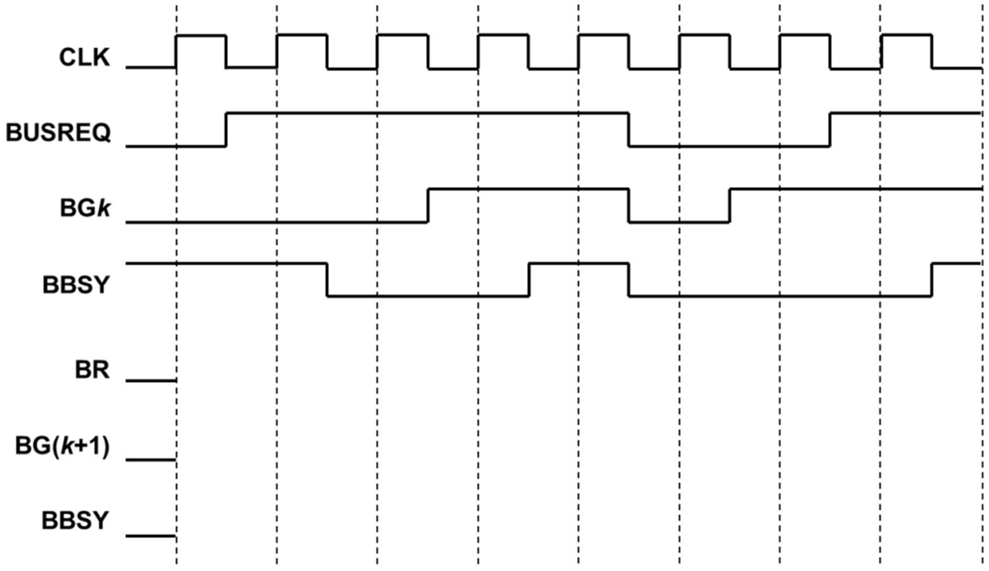
Mealy FSM waveform:

Step 1: Sample input at each rising clock edge and use values to determine state progression

Step 2: Use state information to determine outputs. The outputs will react if there are changes mid clock cycle, so your diagram must reflect this. Make sure to use the output values from the current state!! (The state never changes mid clock cycle)

E.g.





State progression:

Given that first state is idle

1st clock edge: sample was 001, prev state was idle, so period 1 is idle

2nd clock edge: sample was 101, prev state was idle, so period 2 is request

3rd clock edge: sample was 100, prev state was request, so period 3 is request

4th clock edge: sample was 110, prev state was request, so period 4 is granted

5th clock edge: sample was 111, prev state was granted, so period 5 is granted

6th clock edge: sample was 000, prev state was granted, so period 6 is idle

7th clock edge: sample was 010, prev state was idle, so period 7 is pass

8th clock edge: sample was 110, prev state was pass, so period 8 is pass

Outputs:

Period 1 (I): first half of clock cycle input 001 gives output of 000, second half of clock cycle input 101 gives output of 000

Period 2 (R): first half of clock cycle input 101 gives output of 100, second half of clock cycle input 100 gives output of 100

Period 3 (R): first half of clock cycle input 100 gives output of 100, second half of clock cycle input 110 gives output of 100

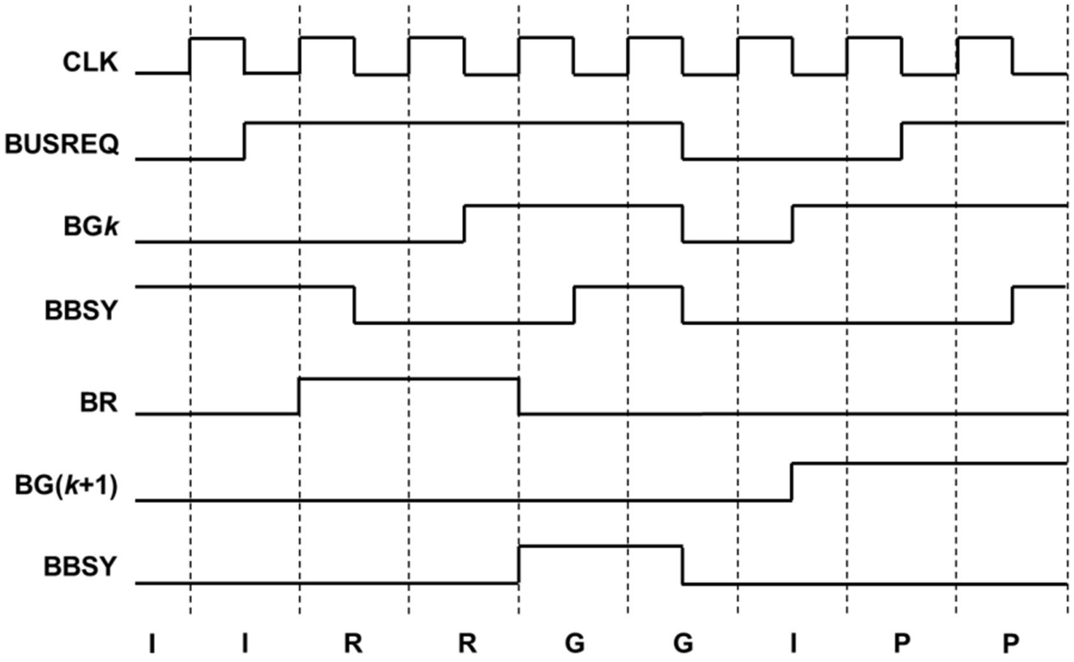
Period 4 (G): first half of clock cycle input 110 gives output of 001, second half of clock cycle input 111 gives output of 001

Period 5 (G): first half of clock cycle input 111 gives output of 001, second half of clock cycle input 000 gives output of 000

Period 6 (I): first half of clock cycle input 000 gives output of 000, second half of clock cycle input 010 gives output of 010

Period 7 (P): first half of clock cycle input 010 gives output of 010, second half of clock cycle input 110 gives output of 010

Period 8 (P): first half of clock cycle input 110 gives output of 010, second half of clock cycle input 111 gives output of 010


CLK
BUSREQ
BG*k*
BBSY
BR
BG(*k*+1)
BBSY
I
I
R
R
G
G
I
P
P