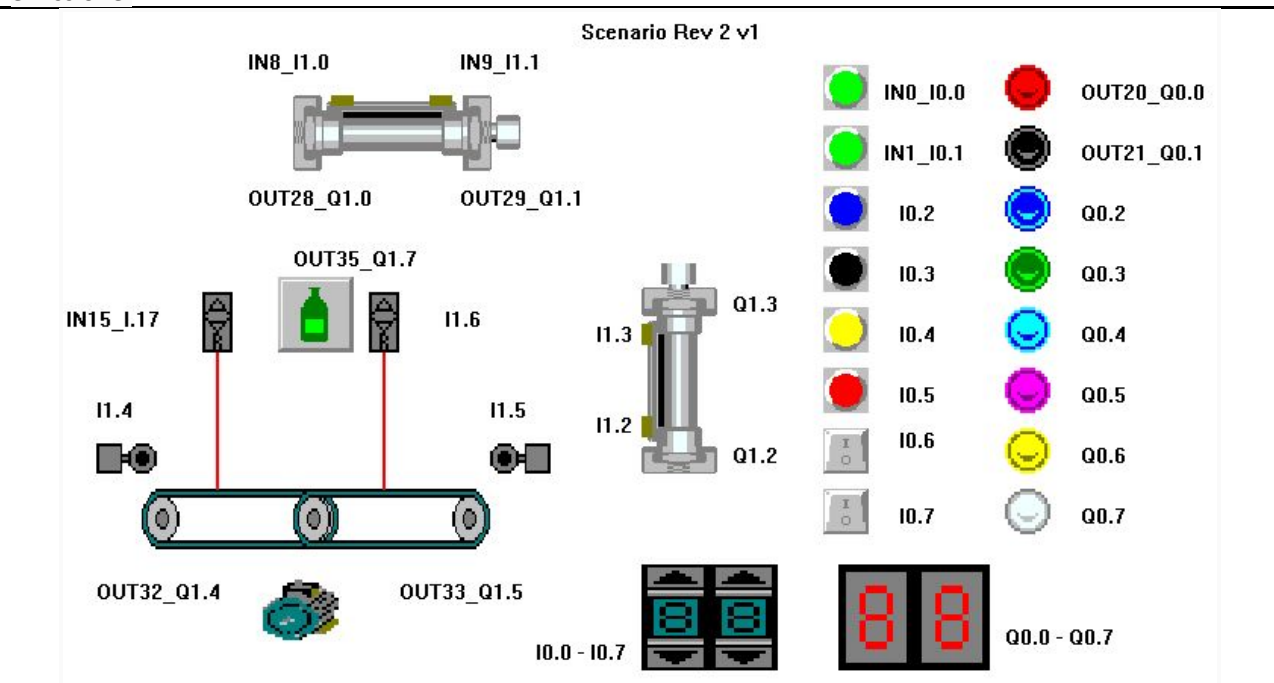


## Programmi per il modulo 1: programmazione di Arduino in ambiente PC\_SIMU con la libreria Scenario.

La revisione 2 del modulo1 utilizza la nuova libreria "HMI.h" di collegamento tra Arduino e PC\_SIMU v2. In questa libreria i pin di Arduino sono simulati e risultano collegati dal pin 0 al 15 con gli indirizzi di ingresso da I0.0 a I1.7 di PC\_SIMU, i pin di uscita da 20 a 35 agli indirizzi di uscita da Q0.0 a Q1.7 di PC\_SIMU.

N.B. La versione di questo documento si compone del numero di revisione (ad es. 2.0) e della data di emissione.



<b>Blink simmetrico</b>	Lampeggia la luce, il tempo Ton è uguale al tempo Toff
<b>Sensori:</b> Nessuno <b>Attuatori:</b> lampadina	Codice: M1Rev2_Blink_simmetrico
<pre> /*  * Turns an LED on for half a second, then off for half a second, repeatedly.  *  * rev. 2 Davide Ciacchi 3/10/18  */ #include &lt;HMI.h&gt;  const int Q00 = 20; // alias per il numero del pin, corrisponde a Q0.0 const int tempoON = 500; // alias per il tempo const int tempoOFF = 500;  void setup() {   /* senza di questa istruzione la libreria non riesce a mettere    in comunicazione Arduino con il programma grafico PC_SIMU */   Serial.begin(19200,SERIAL_8E1);    // con la libreria HMI è inutile qualificare i pin, dato che   // sono assegnati a priori   pinMode(Q00,OUTPUT); } </pre>	

```
void loop() {
  synch();
  digitalWrite(Q00,HIGH);
  delay(tempoON);
  digitalWrite(Q00,LOW);
  delay(tempoOFF);
}
```

**Versione dello stesso programma: senza alias per il numero dei pin, senza commenti, senza righe vuote tra le istruzioni, senza variabili di appoggio.**

```
#include <HMI.h>
void setup() {
  Serial.begin(19200,SERIAL_8E1);
}
void loop() {
  synch();
  digitalWrite(20,1);
  delay(500);
  digitalWrite(20,0);
  delay(500);
}
```

### **Blink variabile**

A partire dal programma Blink per la lampadina in Q0.0, si scriva un programma che permette di modificare il tempo TON con i pulsanti I0.0 e I0.1. Se I0.0 risulta premuto, il tempo TON aumenta di 100 millisecondi, viceversa se I0.1 risulta premuto, il tempo TON diminuisce di 100 millisecondi.

```
/*
 Variante del programma Blink per la lampadina in Q00, permette di modificare
 il tempo TON con i pulsanti I00 e I01. Se I00 risulta premuto, il tempo TON
 aumenta di 100 millisecondi, viceversa se I01 risulta premuto, il tempo TON
 diminuisce di 100 millisecondi;

 rev. 1 Davide Ciacchi 9/12/18

 */

#include <HMI.h>

int tempoON ;
const int tempoOFF = 1000; // questo non cambia
// per chiarezza si da un nome preciso ai pin di comando
const int pinPiu = 0;
const int pinMeno = 1;
const int pinLuce = 20;

void setup() {
  Serial.begin(19200, SERIAL_8E1);
  tempoON =1000;
}

void loop() {
  synch();
  if (digitalRead(pinPiu) == HIGH) {
    tempoON = tempoON + 500;
  }

  // limita il valore massimo
  if (tempoON > 6000) {
    tempoON = 6000;
  }

  if (digitalRead(pinMeno) == HIGH) {
    tempoON = tempoON - 500;
  }

  // limita il valore minimo e
  // garantisce che sia positivo
```

<pre> if (tempoON &lt; 500) {     tempoON = 500; }  digitalWrite(pinLuce, HIGH); delay(tempoON); digitalWrite(pinLuce, LOW); delay(tempoOFF); } </pre>	
<b>Blink snello</b>	Realizza un blink per dimostrare che è possibile leggere lo stato di un piedino impostato come uscita. Legge lo stato precedente dell'uscita e scrive nell'uscita il valore opposto.
<pre> /* Realizza un blink per dimostrare che è possibile leggere lo stato di un piedino impostato come uscita.  Davide Ciacci 30/11/18 */  #include &lt;HMI.h&gt;  const int pinLed = 20;  void setup() {     Serial.begin(19200,SERIAL_8E1);     pinMode(pinLed, OUTPUT);     digitalWrite(pinLed, HIGH); }  void loop() {     synch();     delay(2000);     if (digitalRead(pinLed) == HIGH)         digitalWrite(pinLed, LOW);     else         digitalWrite(pinLed, HIGH); } </pre>	
<b>Blink hardware</b>	Realizza un blink hardware per dimostrare che è possibile leggere lo stato di un piedino impostato come uscita.
<pre> /* Realizza un blink Hardware per dimostrare che è possibile leggere lo stato di un piedino impostato come uscita.  Davide Ciacci 30/11/18 */  const int pinLed = 13;  void setup() {     pinMode(pinLed, OUTPUT);    // obbligatorio per l'hardware non simulato     digitalWrite(pinLed, HIGH); }  void loop() {     delay(1000);     if (digitalRead(pinLed) == HIGH)    // legge lo stato del pin di uscita         digitalWrite(pinLed, LOW);     else         digitalWrite(pinLed, HIGH); } </pre>	
<b>Blink minimo</b>	Realizza un blink hardware utilizzando il minimo del codice possibile, non usa variabili di appoggio, legge direttamente lo stato precedente dell'uscita all'interno della istruzione di scrittura e ne complementa il valore.
<pre> // Davide Ciacci 18/12/18 </pre>	

<pre> void setup() {   pinMode(13, OUTPUT); } void loop() {   delay(1000);   digitalWrite(13, !digitalRead(13)); } </pre>	
<b>Esercizio n.5</b> <b>verifica 3 ATM</b> <b>17-18</b>	Si scriva un programma che, quando è premuto l'interruttore collegato con I0.2, ruota il nastro verso destra per 10 secondi e successivamente ruota il nastro verso sinistra per 5 secondi.
<pre> /*   Si scriva un programma che, quando è premuto l'interruttore   collegato con I0.2, ruota il nastro verso destra per 10 secondi   e successivamente ruota il nastro verso sinistra per 5 secondi.    rev. 1 Davide Ciacchi 9/12/18 */  #include &lt;HMI.h&gt;  const int tempoDestra = 10000; const int tempoSinistra = 5000;  // si genera il numero del pin come somma delle componenti; // offset dell'uscita + byte + bit const int Q14 = 20 + 8 + 4; const int Q15 = 20 + 8 + 5;  void setup() {   Serial.begin(19200, SERIAL_8E1); }  void loop() {   synch();    if (digitalRead(6) == HIGH) {     digitalWrite(Q14, HIGH);     delay(tempoDestra);     digitalWrite(Q14, LOW);     digitalWrite(Q15, HIGH);     delay(tempoSinistra);     digitalWrite(Q15, LOW);   } } </pre>	
<b>And</b>	Frammento di programma che illustra la funzione AND: è NECESSARIO che i pulsanti siano premuti contemporaneamente affinché la lampadina si accenda. Si deve usare obbligatoriamente un pulsante ed un interruttore.
<pre> // se il quinto pulsante è premuto mentre il primo // interruttore è ON, accende la quinta luce if (digitalRead(I04) == HIGH &amp;&amp; digitalRead(I06) == HIGH)   digitalWrite(Q04, HIGH); else   digitalWrite(Q04, LOW); </pre>	
<b>Or</b>	Frammento di programma che illustra la funzione OR: è sufficiente che uno dei due interruttori sia premuto affinché la lampadina si accenda.
<pre> // se il quarto oppure il quinto pulsante // sono premuti accende la terza luce e spegne la quarta if (digitalRead(I03) == HIGH    digitalRead(I04) == HIGH) {   digitalWrite(Q02, HIGH); // accende la luce   digitalWrite(Q03, LOW); // spegne </pre>	

<pre> } else {     digitalWrite(Q03, HIGH);    // accende la luce     digitalWrite(Q02, LOW);    // spegne } </pre>	
<b>Set-Reset</b>	<p>se si preme il pulsante di Set la lampadina si accende e resta accesa finché non si preme il pulsante di Reset. Si usa un pulsante verde per il set, il rosso per il reset. Si noti che il pin corrispondente a Q0.6 è di uscita ma il suo stato (valore) può essere letto. La stessa cosa vale per Arduino non simulato, un pin impostato come OUTPUT può essere letto, viceversa un pin di INPUT non può essere scritto.</p>
<b>Sensori:</b> Due pulsanti agli indirizzi I0.0 e I0.5  <b>Attuatori:</b> una lampadina in Q06	<p>Codice: M1_R2_Set_reset</p>
<pre> /*     Set Reset      rev. 2 Davide Ciacchi 9/12/18 */ #include &lt;HMI.h&gt;  const int I00 = 0; const int I05 = 5; const int Q06 = 20 + 0 + 6;  void setup() {     Serial.begin(19200, SERIAL_8E1);     digitalWrite(Q06, LOW);    // stato iniziale }  void loop() {      synch();      // legge lo stato dei pulsanti, le variabili sono costanti perché     // non devono essere modificate     const int set = digitalRead(I00);     const int reset = digitalRead(I05);     int luce = digitalRead(Q06);      /*         la luce deve essere accesa se è stato premuto il pulsante SET         oppure se SET non è premuto ma la luce era già accesa in precedenza         e se non è stato premuto il pulsante RESET.     */     if ((set == HIGH    luce == HIGH) &amp;&amp; reset == LOW) {         luce = HIGH;    // accende la luce     }     else {         luce = LOW;    // spegne la luce     }      // aggiorna lo stato dell'uscita copiando il valore della variabile     digitalWrite(Q06, luce); } </pre>	
<p><b>Versione dello stesso programma: senza alias per il numero dei pin, senza commenti, senza righe vuote tra le istruzioni, senza variabili di appoggio.</b></p>	
<pre> #include &lt;HMI.h&gt; void setup() {     Serial.begin(19200, SERIAL_8E1);     digitalWrite(26, LOW); } </pre>	

<pre> } void loop() {   synch();   if ((digitalRead(0) == HIGH    digitalRead(26) == HIGH) &amp;&amp; digitalRead(5) == LOW)     digitalWrite(26,HIGH);   else     digitalWrite(26,LOW); } </pre>	
<b>Mirror</b>	Il valore degli ingressi viene copiato sulle uscite, usa quattro modi diversi per fare la stessa cosa.
<b>Sensori:</b> Tutti quelli collegati agli ingressi I00-I07 <b>Attuatori:</b> Tutti quelli collegati alle uscite Q00-Q07	
<pre> /*   Mirror   Riflette lo stato degli ingressi sulle uscite   rev. 1 Davide Ciacci 9/12/18 */  #include &lt;HMI.h&gt;  // vari metodi per definire valori costanti const int I00 = 0, I01 = 1, I02 = 2, I03 = 3;  #define I04 4 #define I05 5  const int I06 = 6; const int I07 = 7;  #define Q00 20 #define Q01 21 #define Q02 22 #define Q03 23 #define Q04 24 #define Q05 25 #define Q06 26 #define Q07 27  void setup() {   Serial.begin(19200, SERIAL_8E1); }  void loop() {   synch();    // non usa variabili di appoggio   // scrive cose diverse in base al valore letto   if (digitalRead(I00) == HIGH )     digitalWrite(Q00, HIGH);   else     digitalWrite(Q00, LOW);    // usa una variabile per ciascuna coppia entrata-uscita   const int valore1 = digitalRead(I01); </pre>	

<pre>digitalWrite(Q01, valore1); const int valore2 = digitalRead(I02); digitalWrite(Q02, valore2);  // usa la stessa variabile per più coppie entrata-uscita // la variabile utilizzata non può più essere costante int valore ; valore = digitalRead(I03); digitalWrite(Q03, valore); valore = digitalRead(I04); digitalWrite(Q04, valore);  // non usa variabili digitalWrite(Q05, digitalRead(I05)); digitalWrite(Q06, digitalRead(I06)); digitalWrite(Q07, digitalRead(I07)); }</pre>	
<b>Esercizio</b>	Se è premuto il pulsante I00 accende le lampadine collegate a Q00, Q02 e Q03.
<b>Esercizio</b>	Se è premuto il pulsante I00 accende le lampadine in Q00 e Q07. Se è premuto il pulsante in I01 accende la lampadina in Q07. Se è premuto il pulsante I02 spegne tutte le lampadine.
<b>Esercizio</b>	Se è premuto il pulsante I00 accende le lampadine in Q00 e Q07. Se è premuto il pulsante in I01 spegne la lampadina in Q07. Se è premuto il pulsante I02 spegne la lampadina in Q00.
<b>Esercizio n.1 verifica 3 ATM 17-18</b>	All'avvio del programma la lampadina collegata con Q0.0 è accesa. Se viene premuto il pulsante collegato con I0.0 si accendono le lampadine collegate a Q0.2 ed a Q0.3, mentre quella collegata a Q0.0 si spegne. Al rilascio del pulsante la lampadina collegata con Q0.2 si spegne mentre quella collegata con Q0.3 resta accesa.
<pre>/* All'avvio del programma la lampadina collegata con Q00 è accesa. Se viene premuto il pulsante collegato con I00 si accendono le lampadine collegate a Q02 ed a Q03, mentre quella collegata a Q00 si spegne. Al rilascio del pulsante la lampadina collegata con Q02 si spegne mentre quella collegata con Q03 resta accesa.  rev. 2 Davide Ciacchi 9/12/18 */  #include &lt;HMI.h&gt;  void setup() {   Serial.begin(19200, SERIAL_8E1);   digitalWrite(20, HIGH); }  void loop() {   synch();    if (digitalRead(0) == HIGH) {     digitalWrite(22, HIGH);     digitalWrite(23, HIGH);     digitalWrite(20, LOW);   }   else {     digitalWrite(22, LOW);   } }</pre>	
<b>Pulsanti esercizio A</b>	Se viene premuto il pulsante I0.0 si accendono le lampadine collegate a Q0.0, Q0.2 e Q0.3. Al rilascio del pulsante le lampadine si spengono

<pre> /*   Se viene premuto il pulsante I00 si accendono le   lampadine collegate a Q00, Q02 e Q03. Al rilascio   del pulsante le lampadine si spengono.   rev. 1 Davide Ciacci 20/11/17 */  #include &lt;HMI.h&gt;  void setup() {   Serial.begin(19200, SERIAL_8E1); }  void loop() {   synch();    if (digitalRead(0) == HIGH) {     digitalWrite(20, HIGH);     digitalWrite(22, HIGH);     digitalWrite(23, HIGH);   }   else {     digitalWrite(20, LOW);     digitalWrite(22, LOW);     digitalWrite(23, LOW);   } } </pre>	
<b>Pulsanti esercizio B</b>	Se viene premuto il pulsante I0.0 si accendono e restano accese le lampadine in Q0.0 e Q0.5. Se viene premuto il pulsante in I0.1 si accende e resta accesa la lampadina in Q0.5. Se viene premuto il pulsante I0.2 tutte le lampadine si spengono e restano spente.
<pre> /*   Se viene premuto il pulsante I00 si accendono e restano accese le lampadine   in Q00 e Q05.   Se viene premuto il pulsante in I01 si accende e resta accesa la lampadina in Q05.   Se viene premuto il pulsante I02 tutte le lampadine si spengono e restano spente.    rev. 1 Davide Ciacci 9/12/18 */  #include &lt;HMI.h&gt;  void setup() {   Serial.begin(19200, SERIAL_8E1); }  void loop() {   synch();    if (digitalRead(0) == HIGH) {     digitalWrite(20, HIGH);     digitalWrite(25, HIGH);   }   if (digitalRead(1) == HIGH) {     digitalWrite(25, HIGH);   }    if (digitalRead(2) == HIGH) {     digitalWrite(20, LOW);     digitalWrite(25, LOW);   } } </pre>	
<b>Pulsanti esercizio C</b>	Se viene premuto il pulsante I00 si accendono le lampadine in Q00, Q01 e Q05, al rilascio di I00 la lampadina in Q00 si spegne mentre quelle in Q01 e Q05 restano



	<p>accese. Se viene premuto il pulsante in I01 si spegne la lampadina in Q05, se viene premuto il pulsante in I05 si spegne la lampadina in Q01</p>
	<pre> /* Se viene premuto il pulsante I00 si accendono le lampadine in Q00, Q01 e Q05, al rilascio di I00 la lampadina in Q00 si spegne mentre quelle in Q01 e Q05 restano accese. Se viene premuto il pulsante in I01 si spegne la lampadina in Q05, se viene premuto il pulsante in I05 si spegne la lampadina in Q01.  rev. 1 Davide Ciacchi 9/12/18 */  #include &lt;HMI.h&gt;  void setup() {   Serial.begin(19200, SERIAL_8E1); }  void loop() {   synch();    if (digitalRead(0) == HIGH) {     digitalWrite(20, HIGH);     digitalWrite(21, HIGH);     digitalWrite(25, HIGH);   }   else {     digitalWrite(20, LOW);   }    if (digitalRead(1) == HIGH) {     digitalWrite(25, LOW);   }    if (digitalRead(5) == HIGH) {     digitalWrite(21, LOW);   } } </pre>
<b>Toggle</b>	<p>Ogni volta che viene premuto il pulsante la lampadina cambia di stato, cioè se era accesa prima della pressione del pulsante si spegne, viceversa si accende. La commutazione avviene solo sul fronte di salita dell'ingresso, cioè solo alla pressione del pulsante, una sola volta.</p> <p>Ad ogni esecuzione della funzione "loop" legge lo stato del tasto di ingresso e lo confronta con lo stato che aveva in precedenza, se lo stato non è cambiato esce dalla funzione "loop" che verrà eseguita successivamente. Se viceversa lo stato del piedino di ingresso è cambiato, anzitutto lo registra nello stato precedente, poi guarda se il nuovo stato è uno stato alto. In questo caso si è verificato un fronte di salita dal valore basso al valore alto e quindi modifica il valore dell'uscita. Se viceversa il valore corrente è basso vuol dire che c'è stata una variazione dello stato dell'ingresso dall'alto al basso, cioè il fronte di discesa che non ci interessa.</p>
<b>Sensori:</b> Un pulsante <b>Attuatori:</b> una lampadina	
	<pre> /* Toggle  rev. 2 Davide Ciacchi 9/12/18 compatibile con HMI.h rev. 2 Davide Ciacchi 10/11/17 corretto livello pulsante premuto </pre>

```

rev. 1 Davide Ciacchi 5/11/17
*/

#include <HMI.h>

int tastoPrima;          // ricorda lo stato del tasto

void setup() {
  Serial.begin(19200, SERIAL_8E1); // init librerie
  tastoPrima = LOW;           // init variabili globali
  digitalWrite(20, LOW);      // init hardware
}

void loop() {
  synch();

  const int tastoAdesso = digitalRead(0); // legge lo stato del pulsante
  int luce = digitalRead(20);             // legge lo stato precedente della luce

  if (tastoAdesso == tastoPrima) { // non è cambiato niente
    return;                       // altro giro
  }

  // qui è cambiato il tasto
  tastoPrima = tastoAdesso; // per cominciare aggiorna la variabile globale

  if (tastoAdesso == HIGH) { // è stato premuto

    // scambia lo stato della variabile luce
    if (luce == HIGH) {
      luce = LOW;
    }
    else {
      luce = HIGH;
    }
    digitalWrite(20, luce); // cambia concretamente la luce
  }
}

```

<b>Esecizi:</b>	●	1) Ogni volta che si preme il pulsante I10 viene incrementata di uno la variabile conteggio.
	● ● ●	2) All'avvio del programma viene emessa una bottiglia e il nastro si muove verso sinistra, quando la bottiglia ingaggia i finecorsa il verso di movimento del nastro cambia. Conta il numero di volte che la bottiglia passa sotto al sensore I16
	● ● ●	3) Come il precedente ma conta il numero di volte che la bottiglia passa sotto al sensore I16 da sinistra a destra.

<b>Funzione "displayNum"</b>	Questa funzione permette di visualizzare sul pannello un numero decimale su due cifre. Ciascun display accetta un valore codificato BCD
----------------------------------	---

<b>Video</b>	<a href="#">prova funzione displayNum</a>
--------------	---

```

#include <HMI.h>
int conteggio = 0;

//-----
void setup() {
  Serial.begin(19200, SERIAL_8E1);
}

```

```
void loop() {
  displayNum(conteggio);
  conteggio = conteggio + 1;
  delay(200);
}
//-----
void displayNum(const int val) {
  // il valore di ingresso viene limitato a due cifre
  int valore = val % 100;
  // sul display di sinistra si visualizzano le decine
  // in codice BCD sugli indirizzi Q0.3-Q0.0
  const int sinistra = valore / 10;

  // converte la cifra decimale in codice BCD
  digitalWrite(23, sinistra & 8);
  digitalWrite(22, sinistra & 4);
  digitalWrite(21, sinistra & 2);
  digitalWrite(20, sinistra & 1);

  // sul display di destra si visualizzano le unità
  // in codice BCD sugli indirizzi Q0.7-Q0.4
  const int destra = valore % 10;

  digitalWrite(27, destra & 8);
  digitalWrite(26, destra & 4);
  digitalWrite(25, destra & 2);
  digitalWrite(24, destra & 1);
}
```

### Conteggio sul fronte

Ad ogni pressione del pulsante in I0.0 viene incrementata la variabile di conteggio. Viene utilizzato il meccanismo del fronte positivo in modo di contare una sola volta, indipendentemente dalla durata della pressione. Questo meccanismo è necessaria tutte le volte che l'azione di innesco del conteggio (**trigger**) non ha valore istantaneo ma dura per un certo tempo, quindi l'evento verrebbe contato più volte.

```
/*
  ad ogni pressione del pulsante incrementa
  la variabile di conteggio
  rev. 2 Davide Ciacci 11/12/18
*/

#include <HMI.h>

int tastoPrima;      // ricorda lo stato del tasto
int conteggio;       // accumula il numero

void setup() {
  Serial.begin(19200, SERIAL_8E1);      // init librerie
}

// init variabili globali
tastoPrima = LOW;
conteggio = 0;

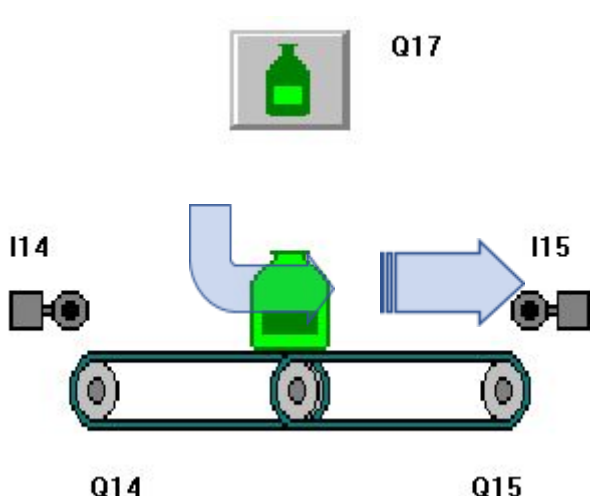
void loop() {
  synch();

  const int tastoAdesso = digitalRead(0);    // legge il tasto

  if (tastoAdesso == tastoPrima) { // non è cambiato niente
    return;                          // altro giro
  }
}
```

<pre>// qui è cambiato il tasto tastoPrima = tastoAdesso;    // per cominciare aggiorna la variabile globale  if (tastoAdesso == HIGH) { // è stato premuto     conteggio = conteggio + 1; } displayNum(conteggio); }</pre>	
<b>Conteggio continuo</b>	<p>Quando il pulsante all'indirizzo I0.0 è premuto viene incrementata la variabile di conteggio.</p> <p>È il meccanismo complementare a quello del conteggio sul fronte, è un meccanismo necessario nei controlli del tipo di quello del volume: si vuole che il conteggio aumenti quando l'operatore continua a mantenere premuto il pulsante.</p>
<pre>/*   alla pressione del pulsante incrementa la variabile di conteggio   rev. 1 Davide Ciacchi 11/12/18 */  #include &lt;HMI.h&gt;  int conteggio;          // accumula il numero  void setup() {     Serial.begin(19200, SERIAL_8E1);          // init librerie      // init variabili globali     conteggio = 0; }  void loop() {     synch();      if (digitalRead(0) == HIGH)    // legge il tasto         conteggio = conteggio + 1;     }     displayNum(conteggio); }</pre>	
<b>Conteggio progressivo</b>	<p>Realizza tutte le funzionalità tipiche di un controllo del volume.</p> <p>Quando è premuto il pulsante in I0.0 il conteggio aumenta sino a 99, quando è premuto il pulsante I0.1 il conteggio diminuisce sino a zero.</p> <p>Per limitare il valore del conteggio viene prima impostato il nuovo valore, poi lo si confronta con il limite superiore o inferiore. Se è stato superato il limite si prende come valore attuale quello del limite.</p> <p>Se il pulsante viene mantenuto premuto la velocità di crescita o di decremento aumenta, per realizzare questo meccanismo di progressione il nuovo valore non viene incrementato sempre della stessa quantità, ma del contenuto di una variabile che a sua volta viene incrementata se il tasto risulta premuto con continuità, viceversa se il tasto viene rilasciato l'incremento torna al valore iniziale unitario.</p> <p>Nella pratica se entrambi i tasti sono non premuti, ed anche se sono premuti entrambi, si riporta l'incremento all'unità, viceversa ogni volta che il tasto è premuto, ed era premuto anche prima, si incrementa di 1 , così l'incremento va sempre crescendo.</p> <p>Inoltre il valore dell'incremento ha il segno, positivo per aumentare e negativo per diminuire, in questo modo si gestisce la situazione in cui si tiene premuto un tasto e</p>

	<p>poi si preme il tasto opposto, in questo caso l'incremento viene riportato al minimo in quanto si cambia direzione.</p> <p>Il conteggio non parte da zero, bensì da un valore intermedio.</p> <p>Questo tipo di funzionalità può essere utilizzata anche per le operazioni di Jog in cui una macchina, fatta funzionare in modalità manuale, viene posta manualmente in posizione tramite un joystick.</p>
	<pre> /*   realizza la funzionalità del controllo del Volume    rev. 2 Davide Ciacchi 13/01/19 introdotto incremento relativo per cambio direzione   rev. 1 Davide Ciacchi 11/12/18 */  #include &lt;HMI.h&gt;  int volume;          // accumula il numero int incremento;      // con segno  void setup() {   Serial.begin(19200, SERIAL_8E1);          // init librerie    // init variabili globali    volume = 20;   incremento = 1; // se si parte da zero una singola pressione non funziona }  void loop() {   synch();   const int alza = digitalRead(0);          // Q0.0   const int abbassa = digitalRead(1);        // Q0.1    // i pulsanti sono uguali in due situazioni, nessun pulsante è premuto   // oppure lo sono entrambi, c'è una discontinuità dell'azione e resetta   // il valore dell'incremento   if (alza == abbassa) {     incremento = 1;     return;   }    if (alza == HIGH) {     if (incremento &lt; 0) // cambio di verso       incremento = 1;     volume = volume + incremento;     incremento = incremento + 1;     if (volume &gt; 99)       volume = 99;     delay(500);          // per il display del numero   }   else{     // è sicuramente l'altro     if (incremento &gt; 0) // cambio di verso       incremento = -1;      volume = volume + incremento;     incremento = incremento - 1;     if (volume &lt; 0)       volume = 0;     delay(500);   } } </pre>
<b>Ping-pong.</b>	<p>All'avvio del programma il nastro trasportatore si avvia immediatamente verso destra. Alla pressione del pulsante di avvio il distributore di bottiglie rilascia una bottiglia che cade sul nastro trasportatore.</p> <p>Quando la bottiglia ingaggia il finecorsa a sinistra il verso di rotazione del motore va</p>

	verso destra, quando ingaggia il sensore di finecorsa di destra il motore inizia il movimento verso sinistra.
<b>Video:</b>	<a href="#">ping pong</a>
<b>Sensori:</b> un pulsante di avvio in I0.0 un sensore di finecorsa a sinistra in I1.4. un sensore di finecorsa a destra in I1.5. <b>Attuatori:</b> un distributore di bottiglie in Q1.7 un motore per la rotazione del nastro verso destra in Q1.4. un motore per la rotazione del nastro verso sinistra in Q1.5	
<pre> /* Pingpong  rev. 2 Davide Ciacchi 11/12/18 */  #include &lt;HMI.h&gt;  const int Q14 = 20 + 8 + 4; const int Q15 = 20 + 8 + 5; const int Q17 = 20 + 8 + 7; const int I00 = 0 ; const int I14 = 8 + 4; const int I15 = 8 + 5;  void setup() {   Serial.begin(19200, SERIAL_8E1);   digitalWrite(Q14, HIGH); // avvia motore } void loop() {   synch();    // ----- lettura dei valori di ingresso   const int versoDestra = digitalRead(Q14);   const int versoSinistra = digitalRead(Q15);   const int limiteDestro = digitalRead(I15);   const int limiteSinistro = digitalRead(I14);   const int start = digitalRead(I00);    // ----- elaborazione   // rilascia la bottiglia   if (start == HIGH)     digitalWrite(Q17, HIGH);   else </pre>	

<pre>digitalWrite(Q17, LOW);  // azionamento del movimento a destra:  // SE ha ingaggiato il finecorsa di sinistra, // OPPURE era già in movimento verso destra // MA NON ha ancora ingaggiato il finecorsa di destra // ALLORA accende il motore nel verso di destra // ALTRIMENTI spegne il motore if ((limiteSinistro == HIGH    versoDestra == HIGH) &amp;&amp; limiteDestro == LOW) {     digitalWrite(Q14, HIGH);    // accende il motore } else {     digitalWrite(Q14, LOW);    // spegne il motore }  // azionamento del movimento a sinistra if ((limiteDestro == HIGH    versoSinistra == HIGH) &amp;&amp; limiteSinistro == LOW) {     digitalWrite(Q15, HIGH);    // accende il motore } else {     digitalWrite(Q15, LOW);    // spegne il motore } }</pre>		
Esercizi:	●	1) Quando ingaggia il finecorsa di sinistra il nastro cambia verso, non lo fa quando ingaggia il finecorsa di destra.
	●●	2) Cambia senso di marcia quando ingaggia il finecorsa per tre volte di seguito, alla quarta volta non cambia senso di marcia e lascia cadere la bottiglia.
	●●●	3) Aumenta il conteggio solo quando transita sotto al sensore da sinistra a destra.
Esercizio n.4 verifica 3 ATM 17-18	Nello schema il nastro trasportatore è dotato di due motori: quello collegato all'indirizzo Q14 permette di ruotare il nastro verso destra, invece quello collegato all'indirizzo Q15 permette di ruotare il nastro verso sinistra. All'avvio del programma viene emessa una bottiglia e contemporaneamente il nastro si mette in movimento verso destra, quando la bottiglia ingaggia uno dei due finecorsa il verso di movimento del nastro cambia. Il programma conta il numero di volte che la bottiglia passa sotto al sensore collegato con l'indirizzo I16	
Video:	<a href="#">ping pong conta doppio</a>	
<pre>/* All'avvio del programma viene emessa una bottiglia e contemporaneamente il nastro si mette in movimento verso sinistra, quando la bottiglia ingaggia uno dei due finecorsa il verso di movimento del nastro cambia. Il programma conta il numero di volte che la bottiglia passa sotto al sensore collegato con l'indirizzo I16.  rev. 1 Davide Ciacchi 11/12/18 */  #include &lt;HMI.h&gt;  const int Q14 = 20 + 8 + 4; const int Q15 = 20 + 8 + 5; const int Q17 = 20 + 8 + 7; const int I16 = 14 ; const int I14 = 8 + 4; const int I15 = 8 + 5;  int sensorePrima;    // ricorda lo stato del tasto int conteggio;  void setup() {</pre>		

```

Serial.begin(19200, SERIAL_8E1);          // init librerie
sensorePrima = LOW;                        // init variabili globali
conteggio = 0;
digitalWrite(Q17, HIGH); // emette la bottiglia
delay(300); // aspetta un pò
digitalWrite(Q17, LOW);
digitalWrite(Q15, HIGH); // avvia motore
}

void loop() {
    synch();

    // gestisce nastro
    const int versoDestra = digitalRead(Q14);
    const int versoSinistra = digitalRead(Q15);
    const int limiteDestro = digitalRead(I15);
    const int limiteSinistro = digitalRead(I14);

    if ((limiteSinistro == HIGH || versoDestra == HIGH) && limiteDestro == LOW) {
        digitalWrite(Q14, HIGH); // accende la luce
    }
    else {
        digitalWrite(Q14, LOW); // accende la luce
    }
    if ((limiteDestro == HIGH || versoSinistra == HIGH) && limiteSinistro == LOW) {
        digitalWrite(Q15, HIGH); // accende la luce
    }
    else {
        digitalWrite(Q15, LOW); // accende la luce
    }

    // gestisce conteggio
    const int sensoreAdesso = digitalRead(I16); // legge il sensore

    if (sensoreAdesso == sensorePrima) { // non è cambiato niente
        return; // altro giro
    }

    // qui è cambiato il tasto
    sensorePrima = sensoreAdesso; // per cominciare aggiorna la variabile globale

    if (sensoreAdesso == HIGH) { // è stato ingaggiato, fronte di salita
        conteggio = conteggio + 1;
        displayNum(conteggio);
    }
}

```

## Ciclo quadro

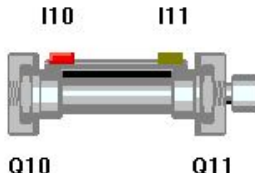
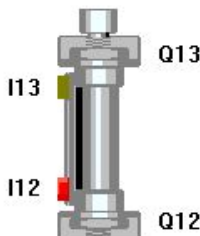
Diagramma delle fasi A+/B+/A-/B-

Alla pressione del pulsante di avvio in I0.0 esce lo stelo del primo cilindro, quando l'uscita dello stelo del primo cilindro è completata inizia l'uscita dello stelo del secondo cilindro. Quando l'uscita è completata inizia il rientro del primo stelo ed in seguito il rientro del secondo stelo.

La funzione "loop" è divisa nelle tre parti canoniche: la prima parte di acquisizione dei valori degli ingressi e delle uscite, cioè si crea l'immagine di ingresso. La seconda parte consiste nell'elaborazione delle variabili in base all'algoritmo, cioè le equazioni univoche del ciclo quadro, viene così prodotta l'immagine di uscita. Infine nella terza parte viene scritta l'immagine di uscita nei pin hardware.

L'algoritmo è diviso in quattro fasi, tante quante sono le equazioni univoche del ciclo quadro. In ciascuna fase viene applicata la tecnica di autoritenuta alla equazione univoca, quindi l'azione sulla valvola del cilindro avviene se è verificata l'equazione univoca, oppure se la valvola era già stata attivata, fintanto che non si è arrivati ad ingaggiare il fine corsa conclusivo.



	Ad ogni esecuzione della funzione "loop" esiste al massimo una fase che è attiva, cioè la sua equazione univoca è verificata.
<p><b>Sensori:</b> un pulsante di avvio, due finecorsa per il primo cilindro, due finecorsa per il secondo cilindro.</p> <p><b>Attuatori:</b> due cilindri bistabili, ciascuno possiede una valvola di uscita ed una valvola di rientro.</p>	 
<pre> /*   rev. 2 Davide Ciacchi 12/12/18 */ #include &lt;HMI.h&gt;  void setup() {   Serial.begin(19200, SERIAL_8E1); }  void loop() {   synch();    //----- LETTURA DI TUTTI I VALORI DI INGRESSO E USCITA-----    // valori di ingresso che vengono solo letti   const int a0 = digitalRead(8);    // I1.0   const int a1 = digitalRead(8 + 1); // I1.1   const int b0 = digitalRead(8 + 2); // I1.2   const int b1 = digitalRead(8 + 3); // I1.3   const int marcia = digitalRead(0); // I0.0    // valori di uscita che vengono prima letti e poi scritti   int Apiu = digitalRead(20 + 8);    // Q1.0   int Ameno = digitalRead(20 + 8 + 1); // Q1.1   int Bpiu = digitalRead(20 + 8 + 2); // Q1.2   int Bmeno = digitalRead(20 + 8 + 3); // Q1.3    //----- ELABORAZIONE DATI E PRODUZIONE USCITE -----    // in ogni istante solo una delle fasi è attiva   // Fase A+   if ((marcia == HIGH &amp;&amp; b0 == HIGH)    Apiu == HIGH) &amp;&amp; a1 == LOW)     Apiu = HIGH;    // cilindro esce   else     Apiu = LOW;    // cilindro fermo    // Fase B+   if ((a1 == HIGH    Bpiu == HIGH) &amp;&amp; b1 == LOW)     Bpiu = HIGH;    // cilindro esce   else     Bpiu = LOW;    // cilindro fermo    // Fase A- </pre>	

```

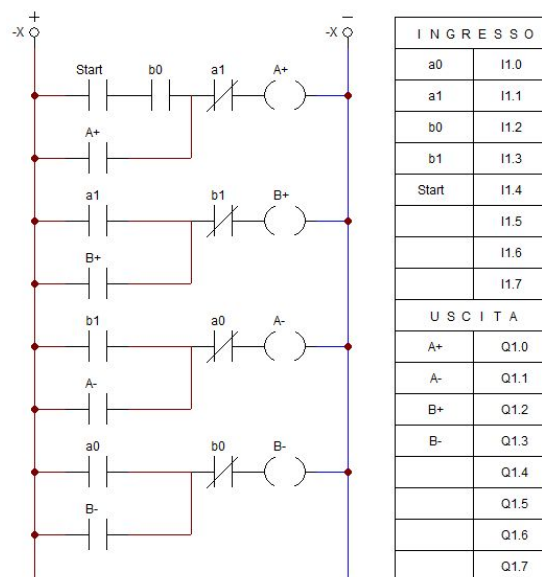
if ((b1 == HIGH || Ameno == HIGH) && a0 == LOW)
    Ameno = HIGH; // cilindro rientra
else
    Ameno = LOW; // cilindro fermo

// Fase B-
if ((a0 == HIGH || Bmeno == HIGH) && b0 == LOW)
    Bmeno = HIGH; // cilindro rientra
else
    Bmeno = LOW; // cilindro fermo

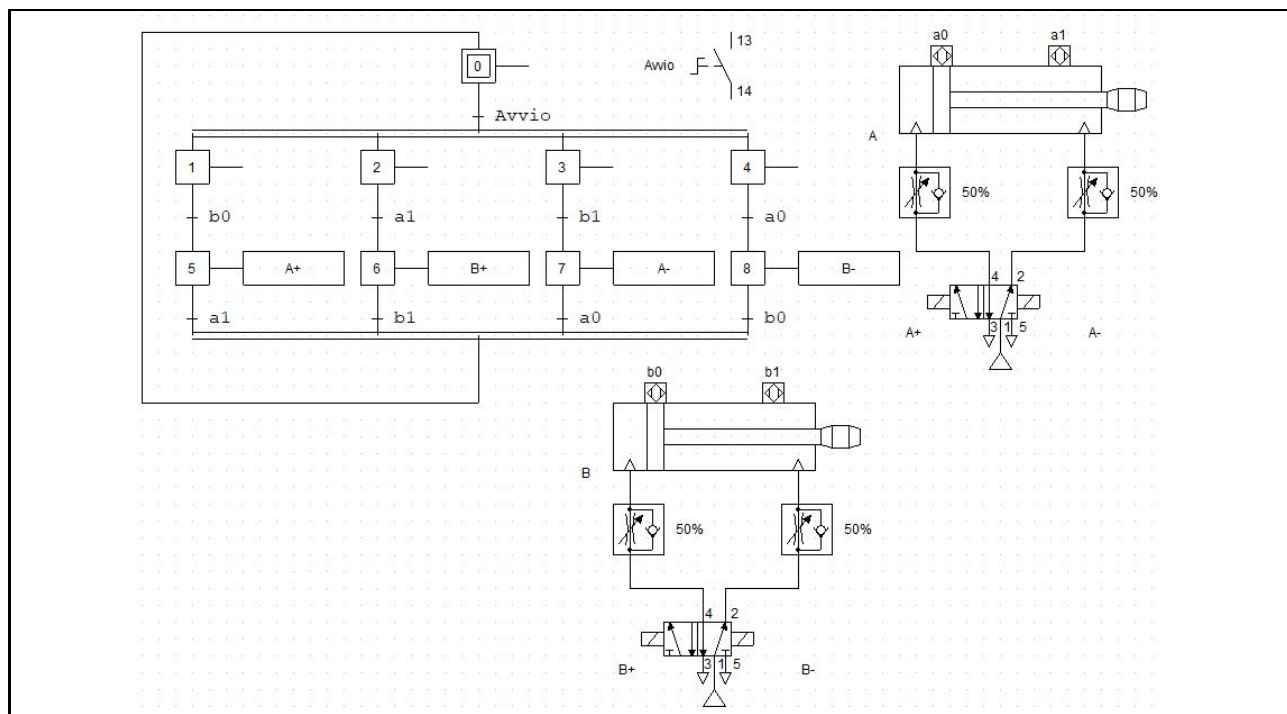
//----- SCRITTURA DELLE USCITE -----
digitalWrite(28, Apiu); // Q1.0
digitalWrite(30, Bpiu); // Q1.2
digitalWrite(29, Ameno); // Q1.1
digitalWrite(31, Bmeno); // Q1.3
}

```

### Ciclo quadro in linguaggio LD (Ladder)



### Ciclo quadro in linguaggio SFC (Grafcet)



<b>Esercizio</b>	Si scriva un programma che realizza il ciclo pneumatico ordinato semiautomatico A-/B+/A+/B-.
<b>Ciclo quadro completo</b>	<p>Il ciclo quadro dell'esercizio precedente viene completato con un insieme di funzionalità che caratterizzano le macchine reali. Il pulsante di avvio collegato all'indirizzo I0.0 inizia il ciclo automatico, quindi il ciclo si ripete finché l'operatore non preme il pulsante di arresto collegato all'indirizzo I0.2, nella pratica il ciclo già iniziato viene completato e non ne viene iniziato un nuovo.</p> <p>Oltre al ciclo automatico è possibile anche il ciclo semi-automatico che viene iniziato premendo il pulsante collegato all'indirizzo I0.1. Per realizzare una funzionalità che somigli in tutto a quella di una apparecchiatura reale, occorre prevedere anche il pulsante di emergenza, collegato all'indirizzo I0.5, quando viene premuto il pulsante di emergenza la macchina si ferma e può ripartire soltanto riprendendo di nuovo il pulsante avvio.</p> <p>Per completare il quadro delle funzionalità di una macchina reale è necessaria anche la modalità manuale (JOG), che viene utilizzata in particolari situazioni di manutenzione e di aggiustaggio. Nel programma seguente la funzionalità manuale è attiva soltanto quando è stato premuto il pulsante di emergenza.</p> <p>Il pulsante in I0.3 controlla il movimento di uscita del pistone selezionato, mentre il pulsante in I0.4 controlla il movimento di rientro. L'interruttore bistabile in I0.6 seleziona il pistone soggetto al movimento manuale di Jog, con selettore basso è sottoposto al Jog il pistone orizzontale, viceversa il pistone verticale.</p> <p>Il funzionamento dell'algoritmo di esecuzione del ciclo è basato sul valore di due variabili inMarcia ed inEmergenza. La variabile inMarcia, quando ha valore alto, in seguito alla pressione del pulsante I0.0, determina l'inizio del ciclo successivo, viceversa quando assume valore basso, in seguito alla pressione del pulsante di arresto I0.2, tutte le fasi rimanenti del ciclo possono essere completate ma non riparte la prima fase.</p> <p>La pressione del pulsante in I0.1, per il semi-automatico, fa avviare la prima fase senza cambiare la variabile inMarcia e realizza un ciclo singolo.</p> <p>Il valore basso della variabile inEmergenza è condizione necessaria per poter eseguire ciascuna delle fasi, in seguito alla pressione del pulsante I0.5 la variabile</p>

diventa alta, nessuna fase del ciclo viene eseguita e la macchina è completamente ferma, sino alla pressione del pulsante di avvio in I0.0.

```

/*
  rev. 2 Davide Ciacchi 12/12/18
*/
#include <HMI.h>

const int I00 = 0, I01 = 1, I02 = 2, I03 = 3, I04 = 4, I05 = 5, I06 = 6;
const int I10 = 8, I11 = 9, I12 = 10, I13 = 11;
const int Q10 = 28, Q11 = 29, Q12 = 30, Q13 = 31;
const int Lmarcia = 23, Lemergenza = 20;

int marcia, emergenza;

void setup() {
  Serial.begin(19200, SERIAL_8E1);
  marcia = LOW;
  emergenza = LOW;
}

void loop() {
  synch();

  //----- LETTURA DI TUTTI I VALORI DI INGRESSO E DI USCITA -----

  // valori di ingresso che vengono solo letti
  const int a0 = digitalRead(I10);
  const int a1 = digitalRead(I11);
  const int b0 = digitalRead(I12);
  const int b1 = digitalRead(I13);
  const int avvio = digitalRead(I00);
  const int arresto = digitalRead(I02);
  const int semi = digitalRead(I01);
  const int jogPiu = digitalRead(I03);
  const int jogMeno = digitalRead(I04);
  const int selettoreJog = digitalRead(I06);

  if (emergenza == LOW)
    emergenza = digitalRead(I05);

  // valori di uscita che vengono prima letti e poi scritti
  int Apiu = digitalRead(Q10);
  int Ameno = digitalRead(Q11);
  int Bpiu = digitalRead(Q12);
  int Bmeno = digitalRead(Q13);

  //----- ELABORAZIONE DATI E PRODUZIONE USCITE -----
  // l'avvio resetta anche l'emergenza
  if (avvio == HIGH) {
    marcia = HIGH;
    emergenza = LOW;
  }
  if (arresto == HIGH || emergenza == HIGH)
    marcia = LOW;

  if (emergenza == LOW) {
    // in ogni istante solo una delle fasi è attiva
    // Fase A+
    if (((marcia == HIGH || semi == HIGH) && b0 == HIGH) || Apiu == HIGH) && a1 ==
LOW)
      Apiu = HIGH;    // cilindro esce
    else
      Apiu = LOW;    // cilindro fermo
  }
}

```

```
// Fase B+
if ((a1 == HIGH || Bpiu == HIGH) && b1 == LOW)
    Bpiu = HIGH;    // cilindro esce
else
    Bpiu = LOW;    // cilindro fermo

// Fase A-
if ((b1 == HIGH || Ameno == HIGH) && a0 == LOW)
    Ameno = HIGH;    // cilindro rientra
else
    Ameno = LOW;    // cilindro fermo

// Fase B-
if ((a0 == HIGH || Bmeno == HIGH) && b0 == LOW)
    Bmeno = HIGH;    // cilindro rientra
else
    Bmeno = LOW;    // cilindro fermo

//----- SCRITTURA DELLE USCITE -----
digitalWrite(Q10, Apiu);
digitalWrite(Q12, Bpiu);
digitalWrite(Q11, Ameno);
digitalWrite(Q13, Bmeno);
}
else {    // emergenza attiva, funziona il Jog
    int cilindroPiu, cilindroMeno;
    if (selettoreJog == LOW) // cilindro A
    {
        cilindroPiu = Q10;
        cilindroMeno = Q11;
    }
    else {    // cilindro B
        cilindroPiu = Q12;
        cilindroMeno = Q13;
    }

    if (jogPiu == HIGH)
        digitalWrite(cilindroPiu, HIGH);
    else
        digitalWrite(cilindroPiu, LOW);

    if (jogMeno == HIGH)
        digitalWrite(cilindroMeno, HIGH);
    else
        digitalWrite(cilindroMeno, LOW);
}

//----- segnalazioni
digitalWrite(Lmarcia, marcia);
digitalWrite(Lemergenza, emergenza);
}
```

**Esercizio ciclo  
con fase di  
inizializzazione**

Realizza il ciclo ordinato semiautomatico A-/B+/A+/B-. La posizione di partenza, con con il cilindro A estratto, viene realizzata all'interno del setup.  
Poiché la funzione setup viene eseguita una sola volta, si aspetta che il cilindro sia uscito sino ad ingaggiare il finecorsa a1. Per attendere il completamento dell'uscita del cilindro si utilizza l'istruzione do-while, all'interno del ciclo do-while si ripete l'istruzione synch per comunicare con l'interfaccia grafica.

```
/*
    rev. 2 Davide Ciacchi 16/12/18
    Realizza il ciclo ordinato semiautomatico A-/B+/A+/B-
    Variante con setup che prepara il cilindro A.
*/
```

```
#include <HMI.h>
```

```

const int I00 = 0, I10 = 8, I11 = 9, I12 = 10, I13 = 11;
const int Q00 = 20, Q10 = 28, Q11 = 29, Q12 = 30, Q13 = 31;

void setup() {
  int a1 ;
  Serial.begin(19200, SERIAL_8E1);
  do {
    synch();
    digitalWrite(Q10, HIGH);    // cilindro esce
    a1 = digitalRead(I11);
  } while (a1 == LOW);
  digitalWrite(Q10, LOW);    // cilindro fermo
}

void loop() {
  synch();
  int a0 = digitalRead(I10);
  int a1 = digitalRead(I11);
  int b0 = digitalRead(I12);
  int b1 = digitalRead(I13);
  int Apiu = digitalRead(Q10);
  int Ameno = digitalRead(Q11);
  int Bpiu = digitalRead(Q12);
  int Bmeno = digitalRead(Q13);
  int start = digitalRead(I00);

  if (start == HIGH)
    digitalWrite(Q00, HIGH);    // indica l'inizio dell'esecuzione del ciclo
  if (b0 == HIGH && Bmeno == HIGH)
    digitalWrite(Q00, LOW);    // indica la fine dell'esecuzione del ciclo

  // Fase A-
  if ((start == HIGH && b0 == HIGH) || Ameno == HIGH) && a0 == LOW) {
    digitalWrite(Q11, HIGH);    // cilindro rientra
  }
  else {
    digitalWrite(Q11, LOW);    // cilindro fermo
  }

  // Fase B+
  if ((a0 == HIGH || Bpiu == HIGH) && b1 == LOW) {
    digitalWrite(Q12, HIGH);    // cilindro esce
  }
  else {
    digitalWrite(Q12, LOW);    // cilindro fermo
  }

  // Fase A+
  if ((b1 == HIGH || Apiu == HIGH) && a1 == LOW) {
    digitalWrite(Q10, HIGH);    // cilindro esce
  }
  else {
    digitalWrite(Q10, LOW);    // cilindro fermo
  }

  // Fase B-
  if ((a1 == HIGH || Bmeno == HIGH) && b0 == LOW) {
    digitalWrite(Q13, HIGH);    // cilindro rientra
  }
  else {
    digitalWrite(Q13, LOW);    // cilindro fermo
  }
}

```

<b>Esercizio ciclo sequenziale</b>	Realizza il ciclo non ordinato semiautomatico A+/B+/B-/A-.
------------------------------------	--

```
/*
  rev. 2 Davide Ciacchi 17/12/18
  Realizza il ciclo non ordinato semiautomatico A+/B+/B-/A-
*/

#include <HMI.h>

const int I00 = 0, I10 = 8, I11 = 9, I12 = 10, I13 = 11;
const int Q10 = 28, Q11 = 29, Q12 = 30, Q13 = 31;
int Memoria;    // per il ciclo sequenziale

void setup() {
  Serial.begin(19200, SERIAL_8E1);
  Memoria = LOW;
}

void loop() {
  synch();
  int a0 = digitalRead(I10);
  int a1 = digitalRead(I11);
  int b0 = digitalRead(I12);
  int b1 = digitalRead(I13);
  int Apiu = digitalRead(Q10);
  int Ameno = digitalRead(Q11);
  int Bpiu = digitalRead(Q12);
  int Bmeno = digitalRead(Q13);
  int start = digitalRead(I00);

  // Fase A+
  if ((start == HIGH && Memoria == LOW) || Apiu == HIGH) && a1 == LOW)
    digitalWrite(Q10, HIGH);    // cilindro esce
  else
    digitalWrite(Q10, LOW);    // cilindro fermo

  // Fase B+
  if ((a1 == HIGH && Memoria == LOW) || Bpiu == HIGH) && b1 == LOW)
    digitalWrite(Q12, HIGH);    // cilindro esce
  else
    digitalWrite(Q12, LOW);    // cilindro fermo

  // Fase M+
  if (b1 == HIGH)
    Memoria = HIGH;

  // Fase B-
  if ((Memoria == HIGH || Bmeno == HIGH) && b0 == LOW)
    digitalWrite(Q13, HIGH);    // cilindro rientra
  else
    digitalWrite(Q13, LOW);    // cilindro fermo

  // Fase A-
  if ((b0 == HIGH && Memoria == HIGH) || Ameno == HIGH) && a0 == LOW)
    digitalWrite(Q11, HIGH);    // cilindro rientra
  else
    digitalWrite(Q11, LOW);    // cilindro fermo

  // Fase M-
  if (a0 == HIGH)
    Memoria = LOW;
}
```