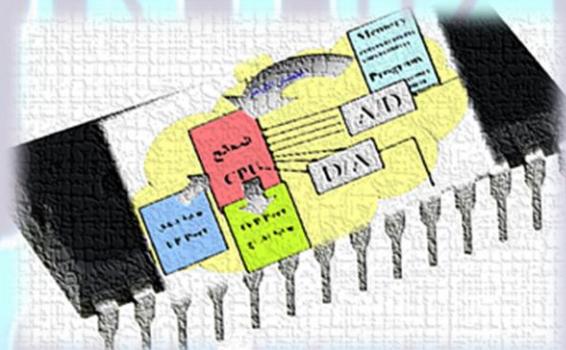


# المتحكمات AVR ... البرمجة والتطبيق

أ. د. محمد ابراهيم العدوى



كلية الهندسة بحلوان جامعة حلوان

نوفمبر 2017

# الإهداء

إلى كل من يحترم لغته ويعتبر بها !!!

## إلى مستخدمي الكتاب

هذا الكتاب منشور تحت رخصة المشاع الإبداعي بشرط عدم الاستغلال التجارى.  
يمكن لأى شخص مشاركة وإعادة توزيع الكتاب مجانا بشرط نسب العمل للمؤلف وعدم  
الاستغلال التجارى.

الكتاب متاح كما ترى للجميع دون أى تكلفة للاستفادة منه على أى وجه سوى الاستغلال التجارى. فرجاء عزيزى القارئ إذا رأيت أنك استفدت منه فلا أطلب منك سوى الدعاء لمؤلفه، والتبرع بما شئت إن شئت، لأى جهة من جهات الخير بنية الثواب للمتبرع والمؤلف.

### المؤلف

أ.د. محمد ابراهيم العدوى

أستاذ متفرغ بكلية الهندسة بحلوان - جامعة حلوان - حلوان - القاهرة

[98eladawy@gmail.com](mailto:98eladawy@gmail.com)

[Mohamed\\_salama01@h-eng.helwan.edu.eg](mailto:Mohamed_salama01@h-eng.helwan.edu.eg)

## استعراض الكتاب

لا شك أننا نعيش الآن في عصر الرقيمات، بل إن شأت قل أننا نعيش في عصر المتحكمات. فكل شيء حولنا أصبح رقمياً بدءاً من فرن الميكرويف، والغسالة، والسيارة، والسيخان، ولعب الأطفال، إلى ساعة اليد، والتليفون النقال (الموبايل) بإمكانياته التي يصعب حصرها الآن. الكثير من الأشياء حولنا أصبحت ذاتية الإدراة، بمعنى أنها لا تحتاج لعامل بشري لإدارتها. فنحن نسمع الآن عن السيارة بدون سائق، والطائرة بدون طيار، والروبوتات بكل أنواعها وإمكانياتها. كل هذه الأشياء التي نعيشها الآن لم تكن حقيقة لولا ثورة الإلكترونيات التي أشعلت شرارتها في بداية الخمسينيات من القرن الماضي بظهور أشباه الوصلات، والتي بلغت أوجها الآن بتصنيع ما يسمى بالأجهزة القابلة للبرمجة programmable microcontroller devices مثل المعالجات الدقيقة microprocessors، والمتحكمات microcontrollers التي أصبحت تلعب دوراً هاماً في حياتنا، بل في أجسامنا، فالكثير منا يحمل بداخل جسمه منظم لضربات قلبه يعمل بوحدة واحدة من هذه الأجهزة. لحسن الحظ فإن التعامل مع هذه المتحكمات لم يعد قاصراً على دارسي الهندسة أو الإلكترونيات، فلقد أصبحنا نرى الآن الكثير من الهواة في الأعمار المختلفة بدءاً من صغار السن في التعليم الأولى والثانوي إلى ما شئت من الأعمار وذوى الخلفيات العلمية المختلفة، ويرجع ذلك إلى ظهور المنظومات البسيطة القائمة على أحد المتحكمات والتي تسهل على المستخدم أو الهاوي التعامل مع هذه المتحكمات من خلال وسط برمجة بسيط يستطيع الهاوي من خلاله إدارة موتور مثلاً أو إضاءة وإطفاء أنوار مختلفة أو حتى إدارة روبوت، والمثال الحى على هذه المنظومات هو منظومة أو لوحة الأردوينو التي شاع استخدامها بين هواة الإلكترونيات هذه الأيام. لقد قام مجموعة من الطلاب الإيطاليين بتصميم وبناء لوحة الأردوينو مستخدمين المتحكم atmega328 الذي هو محور هذا الكتاب لتكون بمثابة وسط برمجة يستطيع الهواة من خلاله التعامل مع إدارة مكونات مادية مثل الموافير وقراءة حساسات مختلفة مثل حساسات الحرارة وغيرها، وهذا هو سبب شيوخ هذه المنظومة بدرجة كبيرة بين الهواة. لقد أخفت منظومة الأردوينو المتحكم الذي يدخلها تماماً عن المستخدم، وأصبح المستخدم يتعامل في طبقة برمجة فوق مستوى المتحكم تماماً بحيث أن المستخدم لا يحتاج على الإطلاق لمعرفة أي تفاصيل عن المتحكم الموجود بداخلها. المطلوب من المستخدم فقط هو معرفة أوامر المنظومة والأطراف المختلفة لها. تماماً مثل برماج تحرير النصوص على الحاسوب والتي منها برنامج الورد من ميكروسوفت على سبيل المثال. في برنامج الورد يقوم المستخدم بكتابة النصوص التي يريد بها من خلال النقر على بعض الأيقونات والقوائم المعروضة أمامه، ويستطيع الطباعة وإرسال نسخ من هذه الملفات إلى من يريد من خلال النقر على بعض الأيقونات أيضاً، وكل ذلك يتم دون أن يكون المستخدم بحاجة إلى معرفة أي تفاصيل عن المعالج الذي هو قلب جهاز الحاسوب الذي يستعمله في ذلك، ولا حتى عن الذاكرة ولا عن أي مكون من مكونات هذا الحاسوب.

من هنا كانت فكرة تقديم هذا الكتاب باللغة العربية لهدفين أساسين: أولهما أن المعالجات والمحكمات أصبحت من المقررات الأساسية في مرحلة البكالوريوس من الدراسة الجامعية في جميع التخصصات الهندسية تقريباً، ومن هنا كان هذا الكتاب يستهدف هذه الفئة من الدارسين حيث تم في هذا الكتاب تغطية معظم الأساسيات النظرية الازمة لطالب الهندسة، بالإضافة طبعاً إلى الجانب العملي من خلال العديد من مشاريع البرمجة الضرورية لإثبات الجانب النظري ووضع الطالب على أرض صلبة يستطيع منها تصميم وبناء أي مشروع يريد. الهدف الثاني من تأليف هذا الكتاب هو أنه تم وضعه بلغة عربية بسيطة وشرح سهل لا يعتمد على كثيراً على الخلفية الدراسية بحيث يستطيع أي هاوي فهم محتوى الكتاب والتطبيق بطريقة سهلة، وبالذات تلك الفئة التي أصبحت أسيرة نظم الأرد Weiner ويريدون الفكاك منها، إلى الرحابة والحرية في العمل على مستوى المحكمات.

ما يؤسف له أن المكتبة العربية والإنترنت بعظمتها تخلو من مادة دراسية أو كتاب بالعربي يغطي هذا المجال إلا المجهود المشكور والرائع والمقدر من إبنتنا العزيز المهندس عبد الله على عبد الله الذي يقوم بجهود رائعة في تعريب الكثير من المواد العلمية، والذي أعزز بأنه كان أحد من قمت بالتدريس لهم في مرحلة البكالوريوس.

الخلفية العلمية الازمة لدراسة هذا الكتاب هي الدراسة بأي لغة بترجمة (وبحذا لو كانت لغة C أو أي واحد من إصداراتها) وبالذات الطرق المختلفة للحلقات وأوامر الشروط والدوال. المطلب الثاني هو كيفية التعامل مع برنامج محاكاة الدوائر الإلكترونية "بروتوس". كل البرنامج والمشاريع التي سنقدمها في هذا الكتاب سيتم تنفيذها بلغة C على برنامج الأتميل استديو، وبعد التأكد من صحتها لغويًا ستنتقل إلى برنامج البروتوس لبناء الدائرة الإلكترونية في هذا الوسط وتنفيذها للتأكد من صحتها إلكترونياً وأن خرج الدائرة يكون وفقاً للدخل الموضوع عليها. بعد هذه الخطوة تكون متأندين أن بناء النظام كدائرة مطبوعة لن تكون به أي مشاكل حيث يمكن البدأ فيه فوراً من يريد ذلك. بالنسبة فإن برنامج البروتوس متاح منه نسخ مجانية على الإنترت ومتاح تلال من الدروس والفيديوهات المختلفة لشرح طريقة التعامل معه.

**الفصل الأول** من الكتاب يغطي مفهوم النظم المدمجة وخصائصها وتعريفاتها المختلفة من قبل العديد من المشغليين في هذا المجال والطريقة المشلى لتدریس كامل عن النظم المدمجة، بالإضافة إلى نظرية تاريخية على تطور صناعة الإلكترونيات بدءاً من الترانزستور حتى ثورة أو عصر المحكمات الذي نعيشها الآن. يغطي الفصل أيضاً العوامل المؤثرة في اختيار معالج أو محكم تبني عليه نظامك المدمج، ثم إذا قررت اختيار أي منها فما هي المواصفات في هذا المعالج أو المحكم الذي ستستخدمه. **الفصل الثاني** تم تحضيره للحديث المفصل عن وحدة المعالجة المركزية Central Processing Unit، حيث هي القلب النابض أو المكون الرئيسي الذي بدونه لا يكون المعالج ولا المحكم. وتم دراسة هذه

الوحدة من حيث الأطراف الخارجية والمكونات الداخلية لها، وهذا الفصل من الضروري قراءته قبل البدأ في باقي الكتاب. **الفصل الثالث** يلقى نظرة موسعة على المتحكمات AVR بصورة عامة من حيث الصفات المشتركة فيها مثل طريقة تفيد الأوامر وأنواع الذاكرة بداخلها. **الفصل الرابع** بدأ بعرض مواصفات المتحكم atmega328 كأحد إصدارات المتحكمات AVR. استعرض الفصل أيضاً بشيء من التفصيل برنامج الأتمل استديو كأحد البرامج التي سنتخدمها في هذا الكتاب لبرمجة ومحاكاة هذا المتحكم. ثم انتقل الفصل إلى إدخال وإخراج البيانات الرقمية من وإلى المتحكم من خلال برنامج يضيء ويطفئ عدد من لمبات البيان (الليدات LEDs). **الفصل الخامس** عبارة مشروعات تطبيقية على إدخال وإخراج البيانات حيث تم فيه استعراض شاشات العرض البلاستيكية LCD وكيفية برمجتها ومواجهتها مع المتحكم. تم أيضاً استعراض استخدام مصفوفة المفاتيح المصغرة keypads وكيفية مواجهتها مع المتحكم، وتم ذلك من خلال عدة مشاريع مختلفة. **الفصل السادس** تكلم عن المقاطعة، من حيث المفهوم العام للمقاطعة ومصادر المقاطعة المختلفة للمتحكم atmega328، مع التركيز في هذا على مقاطعة المتحكم من خلال الأطراف الخارجية، لأن باقي المقاطعات تكون خاصة بالملحقات الأخرى، وسيأتي شرح كل من هذه الملحقات ومقاطعاتها في فصل خاص بكل منها. **الفصل السابع** تناول المحوّل التماثلي الرقمي ADC الموجود داخل المتحكم وكيفية برمجته لتحويل أي إشارة تماثلية مثل إشارة درجة الحرارة أو الصوت أو غيرها إلى الصورة الرقمية والاستفادة بها داخل المتحكم. بالطبع لن يخلو أي مشروع من الحاجة لمثل هذا المحوّل. يحتوي المتحكم atmega328 على ثلاثة موقتات تم شرح كل منها في فصل منفصل. **الفصل الثامن** تناول الموقت صفر بالتفصيل وكيفية برمجته للحصول على أي زمن تأخير صغيراً أو كبيراً، وكيفية الحصول على أشكال موجية مختلفة وأهمها الشكل الموجي ذو العرض المعدل PWM. هذا الموقت يتكون من 8 بت. **الفصل التاسع** تناول الموقت 1 الذي يتكون من 16 بت وكيفية برمجته أيضاً للحصول على أزمنة التأخير والأشكال الموجية المختلفة. **الفصل العاشر** تناول الموقت 2 وهو موقت 8 بت مثل الموقت صفر ويختلف عنه فقط في إمكانية الاستخدام غير المتزامن. الموقتات الثلاثة متباينة تماماً في نظرية العمل وبالرغم من ذلك فقد أفردنا فصلاً خاصاً بكل منهم حتى يكون هناك شرحاً مستقلًا بكل من هذه الموقتات. **الفصل الحادى عشر** عبارة عن فصل تطبيقي تم فيه شرح أكثر أنواع المواتير استخداماً وهي مواتير التيار المستمر، ومواتير المؤازرة servos، ومواتير الخطوة stepper، مع برامج تطبيقية على كيفية مواجهة كل منهم مع المتحكم من أجل التحكم في سرعته واتجاه دورانه. **الفصل الثاني عشر** تناول أحد طرق الاتصالات المتزامنة عبر المسار SPI وخصائص هذا المسار وبرامج تطبيقية عليه. **الفصل الثالث عشر** تناول طريقة من طرق الاتصالات المتسلالية وهي الاتصالات المتسلالية غير المتزامنة عبر الشريحة USART مع شرح مفصل لهذه الطريقة وبرامج تطبيقية عليها. **الفصل الرابع عشر** والأخير تناول طرق حرق فيوزات المتحكم من أجل تأمين البرامج المكتوبة عليه ضد النسخ أو التعديل من قبل الآخرين، وتناول الفصل أيضاً المصادر المختلفة لنبعات تزامن

المتحكم وكيفية توصيل مصادر خارجية لنبضات التزامن. يوجد في نهاية الكتاب **ملحق** خاص بـدوال التعامل مع الشاشات LCD أو مصفوفات المفاتيح التي يمكن النداء عليها وتضمينها في أي برنامج يتعامل مع هذا النوع من الشاشات بدلاً من كتابتها من جديد. يوجد أيضاً في نهاية الكتاب **قاموس** بالكثير من المصطلحات العلمية التي وردت في هذا الكتاب أو التي على صلة به.

محتويات هذا الكتاب مناسبة جداً للتدريس كمقرر دراسي لطلاب الهندسة تخصصات الإلكترونيات والاتصالات والحواسيب، وكليات الحاسوب والمعلومات، وأقسام الميكاترونیات، والهواة المهتمين بالنظم المدمجة. ولقد قمنا بمارسة تدريس هذا المقرر لعدة سنوات على مدار فصل دراسي واحد لمدة ساعتان إسبوعياً كمحاضرة ومعمل ساعتان إسبوعياً لتنفيذ البرامج والمشروعات. المتطلبات السابقة لهذا المقرر هي دراسة مقرر في الدوائر المنطقية ويوجد للمؤلف كتاب بعنوان "الدوائر المنطقية" متاح مجاناً أيضاً على الإنترنت بالإضافة إلى مقرر في البرمجة (وبحذا لو بلغة C أو أحد إصداراتها).

يتقدم المؤلف بالشكر والتقدير للدكتور محمد على ماهر بقسم الهندسة الحيوية الطبية على مساهمته في إخراج النسخة الأخيرة من الكتاب.

## المؤلف

أ.د. محمد ابراهيم العدوى

أستاذ متفرغ بكلية الهندسة بحلوان - جامعة حلوان - حلوان - القاهرة

[98eladawy@gmail.com](mailto:98eladawy@gmail.com)

[Mohamed\\_salama01@h-eng.helwan.edu.eg](mailto:Mohamed_salama01@h-eng.helwan.edu.eg)

# الفصل 1

## مقدمة عن النظم المدمجة

## Introduction to Embedded Systems

**العناوين المضيئة في هذا الفصل:**

- ١- ما هي النظم المدمجة
- ٢- خواص النظم المدمجة
- ٣- خريطة طريق لدراسة النظم المدمجة
- ٤- نظرة تاريخية عن تطور الإلكترونيات
- ٥- وحدة المعالجة المركزية
- ٦- خطوات تصميم النظام المدمج
- ٧- العوامل المؤثرة في اختيار المعالج أو المتحكم
- ٨- ملخص الفصل

### ١-١ ما هي النظم المدمجة؟

## What are embedded systems?

**سنبدأ** بشرح الترجمة المقصودة من التعبير embedded system حتى يكون معناه راسخاً في ذهن القارئ. ترجمة embedded هي المدمج، والمقصود هو نظام إلكتروني مدمج أو مندمج في، أو جزء من، أو مثبت في، أو مختفي في، أو كامن في، نظام آخر كهربائي أو ميكانيكي أكبر منه. من أمثلة ذلك الدائرة التي تحكم في تشغيل غسالة الملابس، عبارة عن نظام إلكتروني مدمج في نظام الغسالة الميكانيكي للتحكم في تشغيله، وأيضاً الدائرة الإلكترونية التي تحكم في تشغيل المصعد، عبارة عن نظام إلكتروني مدمج، أو مثبت، أو جزء من المصعد الذي يعتبر نظام ميكانيكي أكبر. هل هذا يعتبر كافياً كتعريف للنظام المدمج بالمعنى المعروف في هذه الأيام؟ بالطبع لا، فهناك تعريفات كثيرة للنظم المدمجة سنسوق بعضها فيما يلى:

- سنبدأ بالتعريف الموجود في موقع ويكيبيديا المعروف وهو:

"An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints"

وهذا يعني أن النظام المدمج هو نظام حاسب (ونركز هنا على أنه نظام حاسب) له وظيفة محددة داخل نظام ميكانيكي أو كهربائي أكبر، يتحكم فيه في الزمن الحقيقي. يجب ألا نقلق هنا من كلمة نظام حاسب لأن هذا الحاسب يكون إما معالج (ميكروبروسيسور microprocessor)، أو متحكم (ميكروكونترولر microcontroller)، أو معالج للإشارات الرقمية DSP. إن ذلك يعني أننا لن نطلق على النظام الميكانيكي الذي يتم التحكم فيه من خلال دائرة إلكترونية رقمية أو تماثلية لا تحتوى نظام قابل للبرمجة (متحكم أو معالج) أنه نظام مدمج. تحتوى المراجع على الكثير من التعريفات المختلفة الأخرى للنظم المدمجة على حسب وجهات نظر مؤلفى هذه المراجع ونسوق منها ما يلى:

- النظام الكامن عبارة عن نظام تكون وظيفته الأساسية ليست الحساب computation ولكن وظيفته الأساسية هي التحكم (عادة باستخدام متحكم) في النظام الذي تكون كامنة بداخله. وبالتالي فإن ذلك يعني أنه حاسب (متحكم) مدمج داخل نظام ميكانيكي كبير مهمته الأساسية ليست الحساب مثل أجهزة الحاسوب العامة التي نستخدمها في البرمجة بلغات البرمجة المختلفة ونستخدمها في معالجة النصوص وفي إعداد عروضنا التقديمية، ولكن مهمته الأساسية هي التحكم في هذا النظام الميكانيكي فقط، مثلاً في حالة المصعد تكون مهمة النظام المدمج هي التحكم في صعود ونزول المصعد والتوقف عند الأدوار المختلفة وإعطاء إنذارات مختلفة في حالة حدوث أي خلل، وغير ذلك الكثير.

- النظام المدمج عبارة عن نظام إلكتروني يستخدم شريحة ميكروكونتroller، أو شريحة معالج، ولكنها ليست حاسب عام الأغراض في صورة من صور الحاسوبات (سطح المكتب desk top أو حاسب نقال laptop، أو غيرها). وهو نفس التعريف السابق تقريبا.
- النظام الكامن عبارة عن تطبيق يحتوى شريحة واحدة على الأقل قابلة للبرمجة (مثل الميكروكونتroller، أو المعالج، أو شريحة لمعالجة الإشارات DSP)، وهذا التطبيق يستخدم عادة من قبل أشخاص قد لا يعلمون أن هذا النظام يقوم على، أو بداخله، أحد المعالجات أو الميكروكونتroller. وهذا هو التعريف الذى تميل إليه على الرغم من أنه مشابه للتعریفات السابقة ولكنه أضاف أن هذا النظام يتم استخدامه من قبل أشخاص غالباً لا يعلمون أنه يوجد بداخل هذا النظام معالج أو محكم يقوم بهذه المهمة.

## أين توجد النظم الكامنة

معنى ذلك، وبناء على هذه التعريفات، فإنك لو نظرت في نشاطك اليومي ستجد أنك تعيش في وسط عالم من النظم المدمجة تتحكم في حياتنا اليومية بدأً من المنبه الذي يوقظنا من النوم، ثم الفرن الذي ستسخن فيه طعام إفطارك، وماكينة إعداد القهوة، ولا تنسى الساعة التي تلبسها، ثم السيارة أو الحافلة أو القطار الذي ستركتبه للذهاب لعملك (وبالمناسبة فإن وسائل النقل (السيارات) هي الأكثر استخداماً للنظم المدمجة)، ثم المصعد الذي ستتصعد به للأدوار العليا، ولا تنسى أيضاً التليفون النقال (الموبايل) الذي تحمله، والغسالة، والشلاجة، ولعب أطفالك، و.....، وغير ذلك الكثير. ببساطة إننا نعيش في وسط من النظم المدمجة لن نستطيع الفكاك منه إلا إذا ذهبنا إلى الأدغال حتى بدون ملابسنا لأن هناك الآن ما يعرف بالملابس الذكية التي تتحكم في درجة الحرارة المطلوبة، وبالطبع سيكون ذلك من خلال نظام مدمج، ولا أدرى ماذا ستفعل في هذه الحالة إذا كان من سوء حظك أنك ترتدي منظم لضربات قلبك pace maker داخل جسمك، وهذا أيضاً عبارة نظام قائم على نظام مدمج. شكل ١-١ يبين بعضًا من، وليس كل النظم التي من الممكن أن تجده فيها نظاماً مدمجاً.

بناء على هذا الانتشار المهوول للنظم المدمجة في حياتنا فإنه لن يكون غريباً أن نعرف أن حوالي ٩٨٪ من منتجات المحكمات تذهب كلها إلى أغراض النظم المدمجة المختلفة، ومعظم هذه النسبة تذهب إلى صناعة السيارات، حيث كانت هذه الصناعة هي الأسبق في استخدام المعالجات. في عام ١٩٦٨ كانت شركة فولكس واجن الألمانية هي أول شركة تستخدم معالج في حقن الوقود، وبذلك كانت هذه أول شركة تستخدم النظم المدمجة. جاءت بعدها بعدة أعوام شركة فولفو التي استخدمت ١٧ نظاماً مدمجاً في التحكم في العمليات المختلفة في السيارة. بعض موديلات السيارات الحديثة الآن تستخدم ما يفوق المائة محكم أو مائة نظام كامن في التحكم في تشغيلها. هل تتتعجب إذا علمت أن

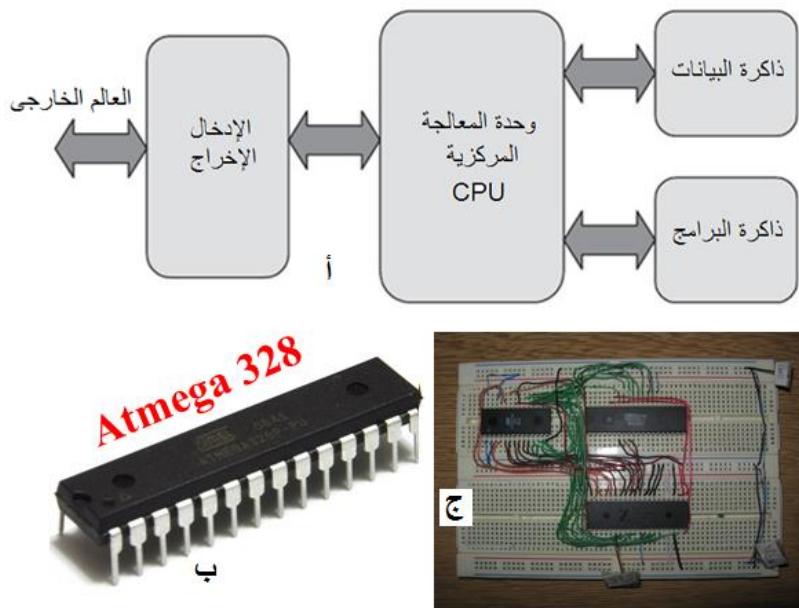
شركة أدياس لصناعة الأحذية قدمت في السنوات الأخيرة حذاء به نظام مدمج يتحكم في راحة اللاعب الذي يلبس الحذاء. يعتقد البعض أنه مع بداية عقد العشرينات (٢٠٢٠) سيحتوى المنزل المتوسط على ما بين ١٠ إلى ١٠٠ متحكم تتحكم في أداء محتوياته.



شكل ١-١ بعض الأنظمة التي من الممكن أن تحتوى نظماً مدمجاً تتحكم في تشغيلها

### باختصار .... ما هو النظام المدمج !

شكل ١-٢أبيّن ر بما صندوقياً مختصرًا لمحتويات أي نظام مدمج. في هذا الشكل تعتبر وحدة المعالجة المركزية Central Processing Unit, CPU بمثابة القلب النابض في النظام المدمج والتي بدونها لا يكون هناك نظاماً مدمجاً. هذه الوحدة قد تكون قائمة بذاتها في صورة معالج يتم توصيل وحدات إدخال وإخراج البيانات وذاكرة البيانات والبرمجة عليها من الخارج، أو أنها قد تكون في داخل متحكم يضم هذه الوحدة بالإضافة إلى بعض وحدات إدخال وإخراج البيانات والذاكرة بنوعيها (البيانات والبرمجة) بالإضافة إلى وحدات أخرى مثل محول للإشارات من الصورة التماثلية إلى الرقمية Serial Analog to Digital Converter, ADC، ومؤقتات Timers، ووسائل للتراسل المتوالى Communication، كل ذلك في شريحة واحدة نطلق عليها المتحكم والذي هو الموضوع الأساسي لهذا الكتاب. وبالطبع يوجد الكثير من هذه المتحكمات من الشركات المختلفة وبإمكانيات مختلفة، وهذا الكتاب كما هو واضح من عنوانه سيركز على أحد منتجات شركة أتميل Atmel وهو المتحكم atmega328 كما في شكل ٢ بـ. أما لماذا هذا المتحكم بالذات من منتجات شركة أتميل الكثيرة، ولماذا شركة أتميل بالذات فسيأتي الحديث عنه لاحقاً.



شكل ٢-١ أ-رسم صندوقى عام لنظام مدمج، ب- النظم على شريحة، ج- النظام على لوحة اختبار

إذن في النهاية سيكون النظام المدمج عبارة عن متحكم مع بعض دوائر المواجهة معه لإدخال واستقبال الإشارات التي سيتم التحكم بها في النظام الميكانيكي الأساسي (المصعد أو الغسالة مثلاً). هذه المكونات من الممكن بناؤها على لوحات اختبار في أثناء المراحل التجريبية للنظام كما في شكل ٢ ج، أو على كارت إلكتروني يتم لحام هذه المكونات عليه في الصورة النهائية، كما هو الحال في كروت التحكم في أي نظام مثل المصعد أو الغسالة.

## ٢-١ خواص النظم المدمجة

### Characteristics of Embedded Systems

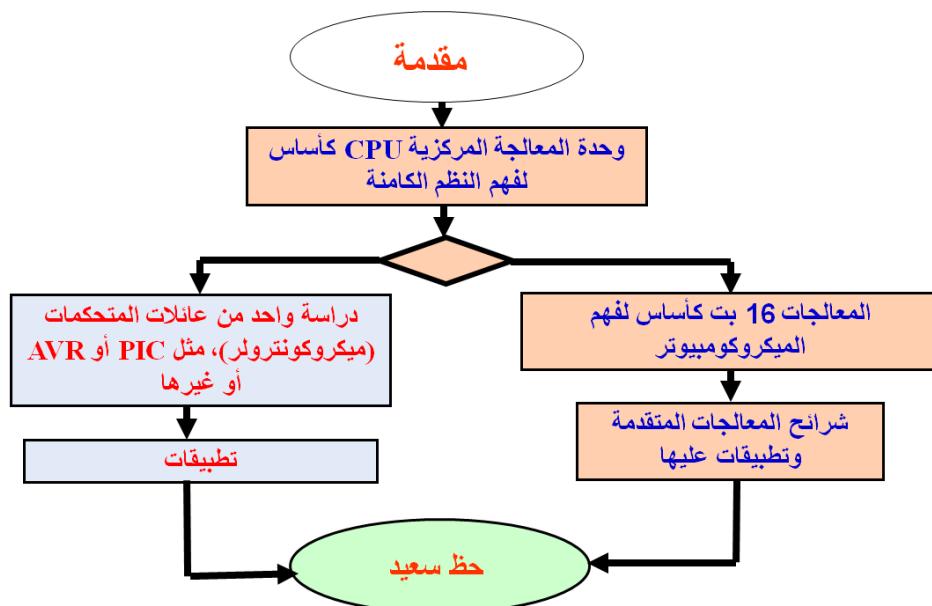
ما سبق يمكّنا أن نضع بعض الخواص العامة لأى نظام مدمج وهي كما يلى:

- ١- النظام المدمج لابد أن يحتوى على حاسب، وهذا الحاسب كما ذكرنا من الممكن أن يكون في صورة معالج أو متحكم أو معالج للإشارات الرقمية DSP.
- ٢- النظام المدمج يتحكم في تشغيل نظام أو جهاز واحد فقط بعينه، ولا يستخدم لأى أغراض أخرى، مثل الحسابات أو الرسم أو حتى التحكم في جهاز آخر.

٣- النظام المدمج يكون ذاتي التشغيل بمجرد توصيله بالقدرة الكهربائية، فلا يحتاج للتدخل البشري لتحميل برنامج مثلاً، أو غير ذلك.

٤- المستخدم للنظام المدمج لا يستطيع الإخبار إذا كان بداخل هذا النظام معالج أو متحكم من عدمه.

بعد أن تعرفنا على النظم المدمجة وخصائصها وكيف أنها منتشرة ومتغلبة في حياتنا اليومية، وحيث أن هذا الكتاب يهدف في النهاية إلى الوصول إلى تصميم نظام مدمج مناسب لأى تطبيق نريده، وحيث أن النظام المدمج لابد أن يحتوى على معالج أو متحكم أو معالج للإشارات الرقمية DSP، فأى واحد من هذه الحاسيبات سندرس هنا. شكل ٣ يبين خريطة طريق مقترحة لأى مقرر أو كتاب يتم إعداده لدراسة النظم المدمجة باستخدام أى واحد من الحاسيبات (معالج، أو متحكم، أو DSP).



شكل ٣-١ خريطة طريق مقترحة لدراسة النظم المدمجة

### ١-٣ خريطة طريق مقترحة لدراسة النظم المدمجة

شكل ٣-١ يوضح هذه الخريطة حيث نلاحظ أن بها مسارين، وكل المسارين مسبق بجزئين أساسيين وهما:

**المقدمة:** وفيها يتم التعريف بالنظم المدمجة وخصائصها وربما يكون من المفيد أن تكون هناك مراجعة تاريخية عن تطور الإلكترونيات يتم فيها إظهار نشأة المعالجات وكيف تطورت هذا التطور السريع، والفرق بين المعالج والمتحكم حتى تختار بينهما على أساس واضح.

**وحدة المعالجة المركزية CPU:** حيث يتم هنا دراسة تفصيلية عن هذه الوحدة الأساسية من حيث أطرافها ووظائف هذه الأطراف ثم التركيب الداخلي لهذه الوحدة من مسجلات عامة وخاصة ووحدة الحساب المنطق. وبالطبع يتم التركيز على وحدة المعالجة المركزية ذات الثمانية برات حيث أنه من خلالها يمكن تقديم الكثير من الأساسيةيات الضرورية لكل من المعالجات والمتحكمات والDSPs.

بعد ذلك تتجه الدراسة إلى أحد المسارين التاليين:

**المسار الأول:** دراسة المعالجات ذات ١٦ بت من حيث الجديد فيها والفرق بينها وبين المعالجات ذات ٨ بت، ولغة التجميع الخاصة بها وكيفية برمجة هذه المعالجات بها وتوصيل وحدات ذاكرة ووحدات إدخال وإخراج للبيانات على هذه المعالجات. والبرامج الفرعية بها ومقاطعتها، ثم تطبيقات على هذه المعالجات ومن المستحسن عمل مشروع تطبيقي عليها. ثم بعد ذلك يتم الانتقال إلى إعطاء فكرة تحصر الجديد في الأجيال المتقدمة من المعالجات. وهذا المسار يحتاج لفصل دراسي كامل.

**المسار الثاني:** وهو يركز على دراسة أحد المتحكمات المتاحة في السوق بالتفصيل من حيث الوظائف الطرفية لها وكميّات الذاكرة الموجودة بها، وأنواعها، وكذلك المكونات الطرفية الموجودة بداخلها مثل محولات الإشارات التماضية إلى رقمية ADC، والمؤقتات Timers، والتواصل التتابعى serial communication، وغير ذلك من المكونات الأخرى على حسب إمكانيات هذا المتحكم وكيفية برمجة هذه المكونات والتعامل معها.

ونحن في هذا الكتاب سنتبع المسار الثاني وسيتم التركيز على المتحكم atmega328، حيث أن هذا المتحكم هو الوحدة الأساسية في لوحة الأردوينو التي شاع استخدامها بكثرة بين الهواة ومن ليست لديهم فكرة عن المعالجات أو المتحكمات. وبالتالي كان هذا هو السبب في التركيز على عائلة متحكمات شركة أتيل والمعروفة بالAVR. وفي الحقيقة يرجع الفضل في شيوع هذه المتحكمات إلى استخدام الأردوينو لها في اللوحات الخاصة بها. ربما يقول البعض إنني متسرس على استخدام لوحات الأردوينو وأستطيع أن أبرمجها لعمل أي تطبيق أريده، فيما هي الحاجة لتعب القلب ودراسة المتحكم بكل تفاصيله. الإجابة على ذلك بسيطة جدا وهي أنه بفرض أنك استخدمت الأردوينو في تصميم أي نظام مدمج وستقوم بوضع لوحة الأردوينو بالكامل في هذا النظام. ألا ترى أنه من الخسارة بمكان أن تضع هذه اللوحة الكاملة المصممة للكثير من الأغراض العامة (ولذلك فإن ثمنها مرتفع) والتي في الغالب لن تستخدم إلا القليل منها في نظامك المدمج وستهمل باقى الوظائف الغير مستخدمة وهذا يمثل إهداراً للكثير من الإمكانيات وبالتالي النقود. في

مقابل ذلك، فإنك بدراسة هذا المقرر ستستخدم فقط المتحكم الذى قد يكون ثمنه أقل من ربع ثمن لوحة الأردوينو الكاملة، وستقوم ببرمجة هذا المتحكم لأداء نفس الوظيفة التى كانت ستؤديها لوحة الأردوينو في النظام المدمج. تخيل أيضاً ماذا سيكون الموقف لو أنك تستنتاج من هذا النظام المدمج كميات كبيرة من هذه الوحدات، بالطبع لن يكون أمامك مفر من استخدام المتحكم وليس لوحة الأردوينو حتى يكون مشروعك اقتصادياً.

المعالج أو المتحكم أو أى شريحة إلكترونية تعتبر في النهاية نظام من الترانزistorات المكثفة بدرجة مهولة جداً على هذه الشريحة، فلنك أن تخيل مثلاً أن أى واحد من المعالجات الحديثة قد يتكون من مائة مليون ترانزistor مبنية في شريحة إلكترونية قد يكون مساحتها الحقيقي حوالي  $3 \times 3$  ميلليمتر. فكيف وصلت الأمور إلى ما نحن عليه الآن؟ إن ذلك يحتاج إلى نظرة تاريخية عن تطور الإلكترونيات حتى نرى كيف وصل الأمر إلى هذا الحد حتى أن أحدهم يقول أنه لو أن صناعة السيارات كانت تتتطور بنفس معدل تطور الإلكترونيات لوصل الأمر أن الإنسان كان سيشغل سيارته ويسير بها إلى القمر فوراً.

## ١-٤ نظرة تاريخية على تطور الإلكترونيات

إن الطفرة الأخيرة التي حدثت في علوم الحاسوب يرجع الفضل فيها أساساً إلى التقدم في علوم الإلكترونيات والتكنولوجيا الحديثة والمتطرفة في تصنيع الدوائر التكاملية Integrated circuits، فما هي الدائرة التكاملية إذن؟ لكن نعرف ما هي الدائرة التكاملية تعالى نرجع إلى الوراء في التاريخ وبالتحديد في عام ١٩٤٩ عندما تم اكتشاف الترانزistor Vacuum Transistor tubes التي كان منها ما يكافئ الترانزistor ومنها ما يكافئ الدايد Diode على سبيل المثال، وكان أى صمام من هذه الصمامات عبارة عن اسطوانة زجاجية مفرغة من الهواء يبلغ قطرها حوالي ثلاثة سنتيمترات وارتفاعها حوالي سبعة سنتيمترات، وكانت هذه الصمامات تحتاج لتشغيلها إلى فرق جهد مستمر d.c عالي يبلغ فوق ٢٠٠ فولت، ولذلك كانت جميع الأجهزة الإلكترونية في هذا الوقت تعرف بـ حجمها، فلنك أن تخيل مثلاً أن جهاز حاسب شخصي من أبسط الأجهزة المعروفة الآن ربما كان يشغل حجرتين كاملتين متوسطي الحجم لو أنه بني بهذه الصمامات.

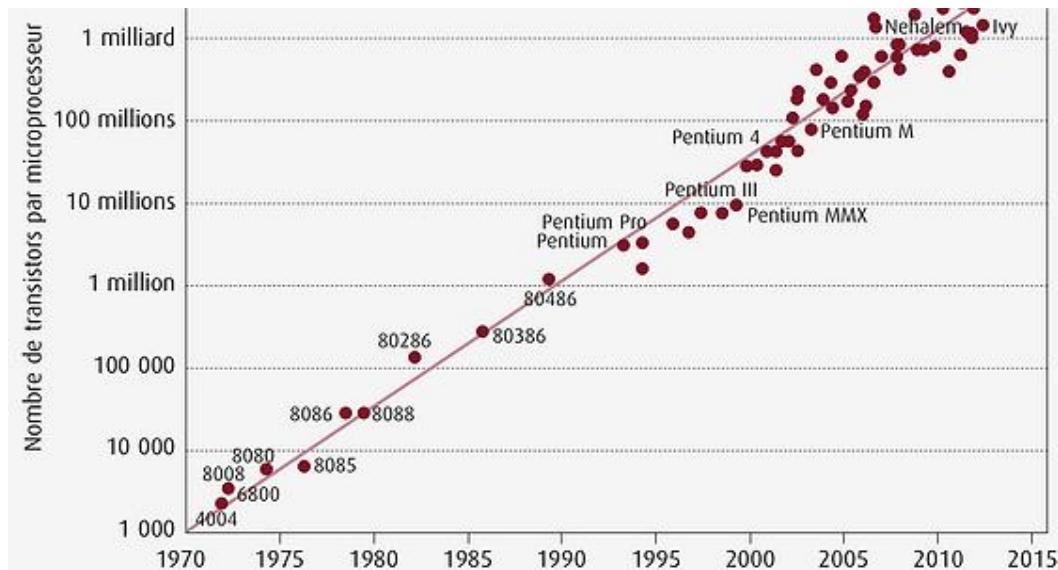
باكتشاف أشباه الموصلات وظهور الترانزistor أخذت أحجام الدوائر الإلكترونية والفراغ الذي تشغله في الانكماش، ومنذ ذلك الحين بدأت عجلة التطور في بناء الدوائر الإلكترونية في الدوران، وأصبح المصممون لا يكتفون ببناء ترانزistor

واحد على نفس شريحة شبه الموصل، ولكنهم بدأوا في وضع أكثر من ترانزistor على نفس القطعة، ثم أضافوا لهذا العدد من الترانزستورات بعض المكونات الأخرى مثل المقاومات والمكثفات، ثم قاموا بتوصيل هذه المكونات مع الترانزستورات الموجودة على نفس الشريحة للحصول على دائرة إلكترونية تؤدي وظيفة معينة، هذه الدائرة الإلكترونية المبنية على شريحة واحدة لأداء هدف أو وظيفة معينة هي ما أطلق عليه بالدائرة التكاملية. في بداية الستينات كان كل ما تمكنت منه التكنولوجيا في ذلك الوقت هو بناء أو تجميع حوالي عشرة ترانزستورات على نفس الشريحة واستخدمت هذه في بناء دوائر البوابات المنطقية مثل بوابة AND وبوابة OR وغيرها وهيئت هذه الدوائر **بدوائر التكامل الصغير (SSI)**.

بعد ذلك أخذت تكنولوجيا بناء الدوائر التكاملية في التطور السريع حيث تمكن المصممون من زيادة كثافة المكونات على نفس الشريحة فظهرت الدوائر ذات التكامل المتوسط (MSI) والتي منها على **Medium Scale Integration (MSI)** سبيل المثال دوائر العدادات counters ومسجلات الإزاحة shift registers والكثير من المكبرات التماضية analog amplifiers المتعددة الأغراض، ولم يقف الأمر عند هذا الحد بل ظهرت بعد ذلك **الدوائر عالية التكامل (LSI)** والتي منها شرائح الذاكرة memory وشرائح المعالجات بجيليها الأول والثانى والتي من其中 المعالج Intel4004 الذى كان يحتوى على ۲۳۰۰ ترانزistor على نفس الشريحة. بالنسبة كانت سرعة (نبضات التزامن) لهذا المعالج ۱۰۸ كيلوهertz ، ومسار البيانات له ۴ بت وظهر في السوق في عام ۱۹۷۱ . لم يقف الأمر عند هذا الحد أيضا بل ظهرت بعد ذلك **الدوائر التكاملية الفائقة التكامل (VLSI)** والتي منها بعض شرائح الذاكرة والأجيال الأخيرة من شرائح المعالجات والتي منها الجيل الثالث والرابع والتي يمثلها المعالجات مثل Intel8080 و Intel8085 و Z80 ومعالجات أخرى كثيرة. هذه المعالجات كان يحتوى الواحد منها على ۶۰۰۰ ترانزistor على نفس الشريحة وكانت سرعتها ۲ ميجا赫تز ومسار البيانات لها ۸ بت وظهرت كلها ابتداء من عام ۱۹۷۴ . لك أن تخيل الآن أن عدد الترانزستورات على الشريحة الواحدة التي لا تتعدي مساحتها المستترمتر المربع الواحد قد فاق عدة ملايين من الترانزستورات على نفس الشريحة، فالمعالج بت يتم Pentium4 مثلا يحتوى على ۱۵ مليون ترانزistor على نفس الشريحة وسرعته تعدد ۳ جيجا赫تز وظهر في الأسواق عام ۲۰۰۰ . لقد وصلت كثافة الترانزستورات هذه الأيام ( ۲۰۱۷ ) وقت كتابة هذا الكتاب حواله ۱۰۰ مليون ترانزistor)، وبعلم الله وحده ما سيأتي لنا به المستقبل القريب وإلى أين سيصل هذا العقل البشري؟ هذه النعمة التي دائما يحاول الإنسان تقليدها ولكنه دائما سيخفق في تصنيعها !!

### قانون موور Moor's law

ينص قانون موور (موور هو مؤسس أحد شركات الإلكترونيات Fairchild) على أن كثافة الترانزستورات على الشريحة الإلكترونية يتضاعف كل 18 شهر إلى سنتين تقريباً، وقد أثبتت مور ذلك برسم العلاقة بين عدد الترانزستورات على مدى 45 سنة تقريباً من عام 1970 حتى عام 2015 ووجد أنها محققة تقريباً وبدقة معقولة كما في شكل ٤-١.



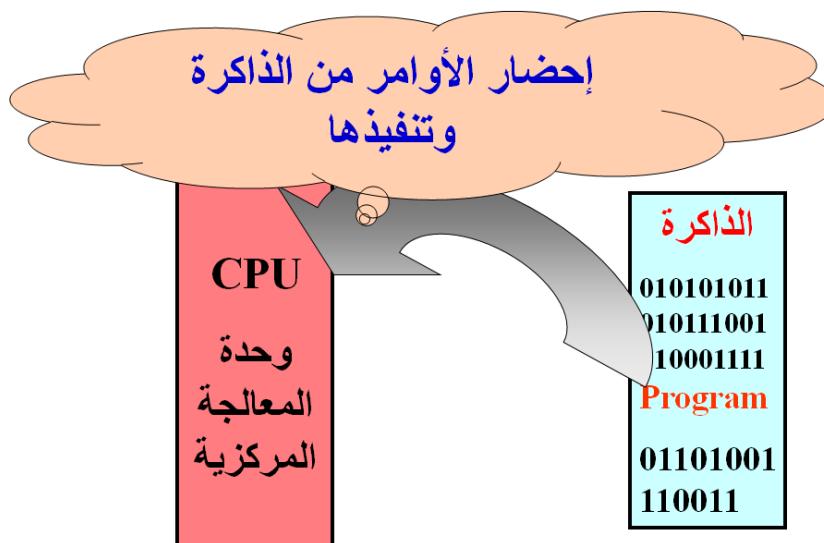
شكل ٤ قانون موور Moor's law

بناء على هذه الكثافة العالية من الترانزستورات والتقدم السريع في هذه الصناعة كانت هناك إحصائية ظريفة نشرت من قبل رابطة مصنعي أشباه الموصلات في أمريكا في عام ٢٠٠٣ تفيد بأنهم يستهدفون أن يكون نصيب الفرد على مستوى العالم من الترانزستورات المنتجة هو بليون ترانزستور في اليوم!!! وهنا يجب علينا نحن سكان العالم الثالث أن نقف وقفه ونسأل أين نصيبنا من هذه الترانزستورات !!!

## ٥-١ وحدة المعالجة المركزية CPU

وحدة المعالجة المركزية CPU هي الأساس لأى وحدة حاسب سواء كان هذا الحاسب معالج أو متحكم أو معالج للإشارات الرقمية DSP. في البداية كانت وحدة المعالجة المركزية تبني باستخدام أكثر من شريحة واحدة. مثلاً شريحة لوحدة الحساب والمنطق، وأخرى للمسجلات، وثالثة لنظام التزامن، وهكذا. بناء هذه المكونات على شريحة واحدة هو ما أطلق عليه فيما بعد المعالج أو الميكروبروسيسور microprocessor. أى أن المعالج هو وحدة معالجة مركزية على شريحة واحدة. من شكل ٤-١ كان أول ظهور لوحدة معالجة مركزية (معالج)

انتشرت واستخدمت في الصناعة في عام ١٩٧٥ تقريباً، حيث قدمت معظم شركات الإلكترونيات المعالج (أو وحدة المعالجة المركزية) الخاصة بها. ففي هذا الوقت أصدرت شركة إنتل منتجها Intel8080 و Intel8085، ثم المعالج Z80 من شركة زايلوج، والمعالج MC6800 من شركة موتورولا، وكلها معالجات ذات ٨ بت.



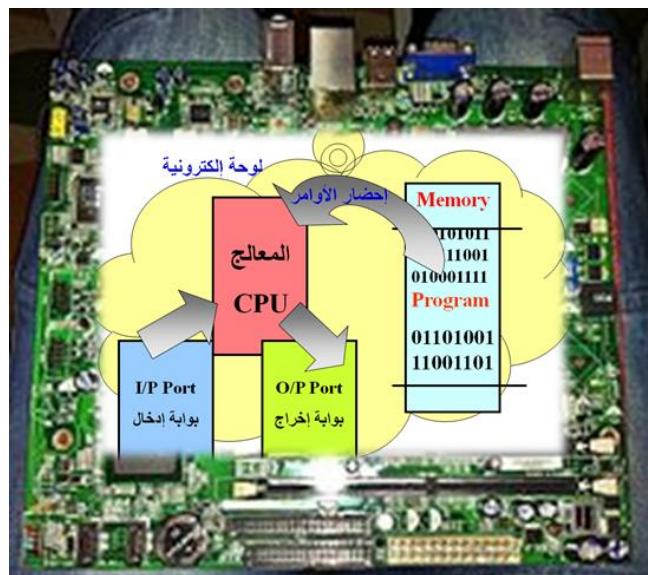
شكل ١-٥ وظيفة وحدة المعالجة المركزية هي إحضار الأوامر من الذاكرة

كما نرى فإن وحدة المعالجة المركزية والمعالج هما مرادفان لنفس الشيء تقريباً. فما هي وظيفة المعالج أو وحدة المعالجة المركزية التي قلبت العالم تكنولوجياً منذ ظهورها الحقيقي في عام ١٩٧٥ تقريباً؟ إننا سنتعجب عندما ندرس وحدة المعالجة المركزية بالتفصيل في الفصل ٢ حيث سنجد أنها من أبسط ما يكون من حيث التركيب والوظيفة. إن الوظيفة الأساسية لوحدة المعالجة المركزية هي إحضار الأوامر من الذاكرة وتنفيذها، الأمر بعد الآخر حتى تصل إلى نهاية البرنامج. أى أن هذه الوحدة لا تحتوى على أماكن لتخزين البرامج أو البيانات. ولذلك فإن المعالج وحده وكوحدة قائمة بذاتها يعتبر عديم الفائدة. ولذلك فإنه لكي يتم الاستفادة من وحدة المعالجة المركزية فإنه لابد من توصيلها مع ذاكرة ووحدات إدخال وإخراج للبيانات، وبذلك تصبح ميكروكمبيوتر، وهو ما سنتعرف عليه في الجزء التالي. شكل ١-٥ يبين رسمياً توضيحاً لوظيفة المعالج أو وحدة المعالجة المركزية.

## الميكروكمبيوتر

كما رأينا فإن المعالج وحدة لا ينفع بشيء، ولذلك لكي يتم الاستفادة منه فلا بد من أن توصل معه ذاكرة يتم فيها تسجيل البرنامج الذي سينفذه المعالج، وأيضاً لتخزين البيانات التي قد يحتاجها هذا البرنامج أو تنتج منه. كذلك فإن

هذا البرنامج قد يحتاج للتعامل مع بوابات إدخال للبيانات يتم من خلالها إدخال بيانات للمعالج، مثل لوحة مفاتيح أو إشارات تمثل متغيرات يتم التحكم فيها عن طريق هذا المعالج. أيضاً قد يحتاج البرنامج إلى بوابات لإخراج البيانات، فقد يحتاج المعالج لعرض بيانات على شاشة عرض أو إخراج إشارة معينة تستخدم في إدارة أحد الحركات. المعالج، والذاكرة، وبوابات الإدخال، وبوابات الإخراج عند تجميعها على لوحة هي ما يسمى الميكروكمبيوتر. وقد تم استخدام مثل هذا الميكروكمبيوتر في الكثير من التطبيقات وأهمها الحاسوبات عامة الأغراض التي توجد الآن بصور مختلفة منها الحاسوب النقال lap top على سبيل المثال. لم يقتصر الأمر على ذلك بل استخدمت هذه الكروت التي تمثل الميكروكمبيوتر في الكثير من أغراض التحكم في العمليات الصناعية وغيرها. شكل ٦-١ يبين رسمياً توضيحاً لهذا النوع من الحاسوبات.



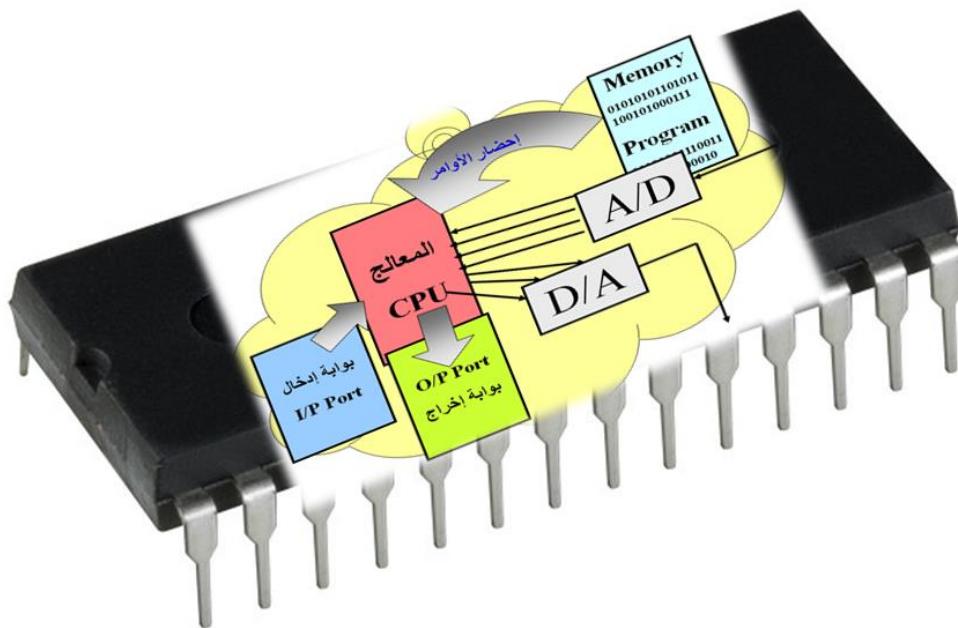
شكل ٦-١ الميكروكمبيوتر على لوحة إلكترونية

### المتحكم (الميكروكونترولر)

إذا تم تجميع كل مكونات الميكروكمبيوتر السابقة من ذاكرة وبوابات لإدخال وإخراج البيانات، مع إمكانية إضافة محول تماثلي إلى رقمي ومحول رقمي إلى تماثلي، ومؤقتات، وإمكانيات للتعامل مع البيانات التتابعية، وغير ذلك الكثير، إذا تم تجميع كل ذلك في شريحة واحدة، فإن هذه الشريحة هي ما يسمى بالمتحكم أو الميكروكونترولر microcontroller الذي هو موضوع الدراسة في هذا الكتاب. شكل ٧-١ يبين رسمياً توضيحاً لهذا المتحكم. من ذلك فإنه يمكننا أن نقول أن المتحكم عبارة عن حاسب كامل على شريحة واحدة.

كما رأينا فإن المتحكم بداخلة وحدة حساب أو حاسب وهي وحدة المعالجة المركزية، بالإضافة إلى بعض المكونات الأخرى، كما أنه يستخدم في التحكم أو تشغيل الأنظمة الكبيرة، وهناك أطراف خارجة منه لإدخال وإخراج البيانات

فقط، فهل هناك ما يمنع من أن نطلق على المتحكم نفسه بأنه نظام مدمج؟ إن هذا هو الواقع فعلاً حيث أن البعض بل ربما الكثير من مستخدمي المتحكمات يطلقون عليه أنه نظام مدمج. ونحن نعتقد أن مثل هذا التعريف بالرغم من صحته إلا أنه يكون قاصراً جداً كتعريف للمتحكم ولا نفضل استخدامه.



شكل ٧-١ رسم توضيحي لمحتويات المتحكم

من كل مasic، يمكننا الآن أن نفرق بين المعالج، والمتحكم، والميكروكومبيوتر. المعالج هو الأساس في تكوين كل من المتحكم والميكروكومبيوتر. الميكروكومبيوتر يتم تصميمه في العادة ليستخدم في أغراض الحساب العامة مثل معالجة النصوص والرسم وتنفيذ البرامج باللغات المختلفة، وأما المتحكم فيتم تصميمه لأغراض التحكم في نظم أخرى كبيرة مثل المصاعد والأدوات المنزلية وفي الأغراض الصناعية المختلفة. شريحة المعالج تحتاج للعديد من الشرائح الأخرى المساعدة مثل شرائح الذاكرة وشرائح لإدخال وإخراج البيانات، وأما المتحكم فلا يحتاج لتوصيل مثل هذه الشرائح من خارجه لأنها موجودة أصلاً بداخله. بناء على ذلك فإن أطراف أو أرجل شريحة المعالج تكون عبارة عن خطوط عناوين وبيانات وتحكم بالإضافة إلى خطوط القدرة، وأما المتحكم فإن معظم أطراfe وإن لم يكن كلها تكون عبارة عن خطوط لإدخال وإخراج البيانات بالإضافة إلى خطوط القدرة وسنرى ذلك بالتفصيل في الفصل القادم الخاص بوحدة المعالجة المركزية CPU.

## ٦- خطوات تصميم النظام المدمج

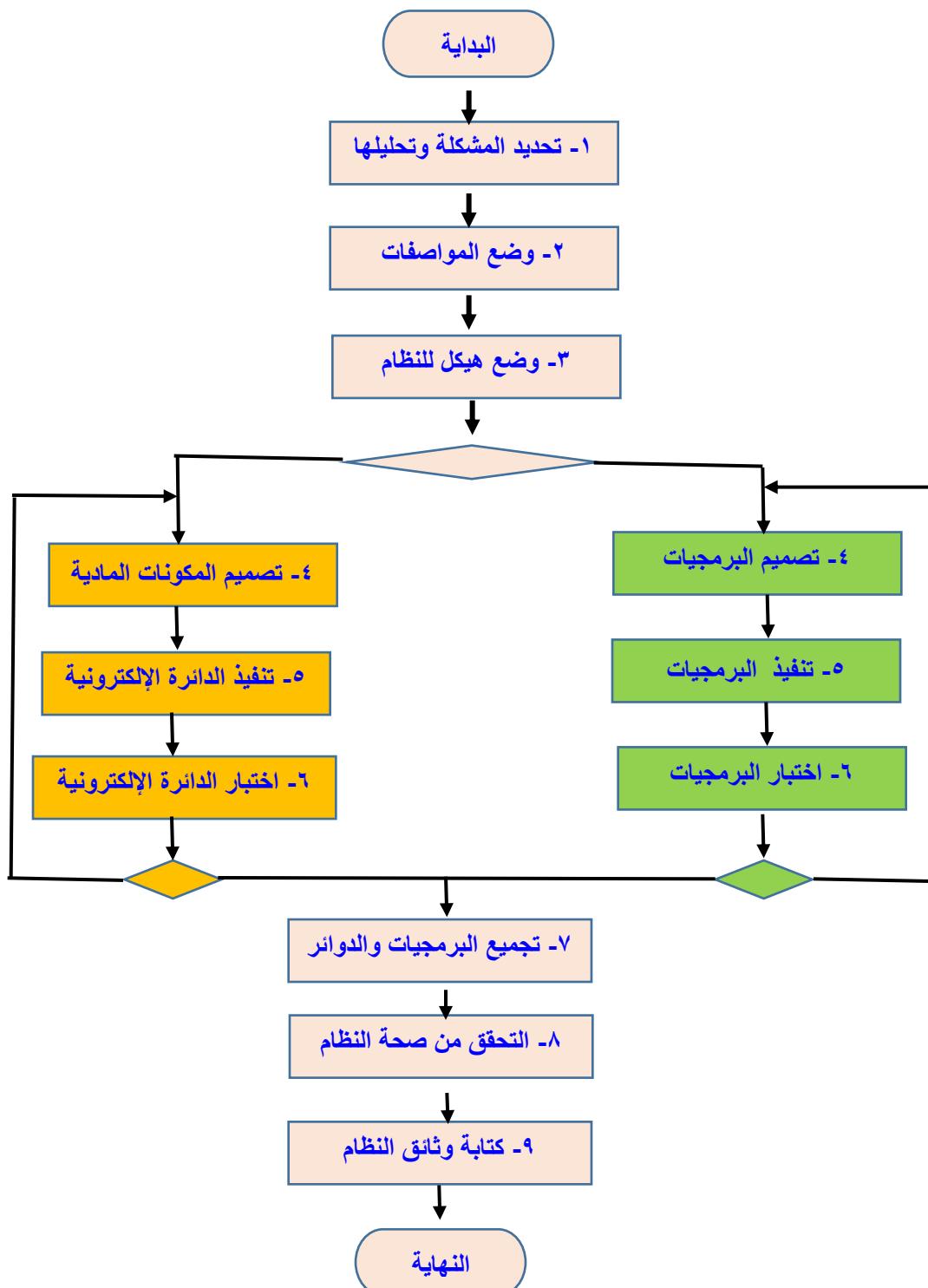
سنرى في هذا الجزء الخطوات العامة التي يتم اتباعها عند تصميم أي نظام مدمج، وبالطبع من الممكن أن تكون هناك اختلافات بسيطة بين نظام وآخر على حسب تعقيد هذه النظم وطريقة المصمم في تنفيذ أفكاره. شكل ٨-١ يبين مقترحاً لمخطط صندوقى أو خريطة سير لتنفيذ هذه الخطوات، وهذه الخطوات يمكن تلخيصها فيما يلى:

### ١- تحديد المشكلة وتحليلها Problem definition and analysis

بفرض أن المشكلة المراد تصميماً لها هي تصميم نظام مدمج يتتحكم في تشغيل غسالة ملابس مثلاً. في هذه الحالة يجب أن نحدد أشواط الغسالة مثل شوط لسحب المياه، وشوط للغسيل، وآخر لعصر الملابس، وربما يكون هناك شوط آخر لإضافة ماء ثم العصر، وزمن كل شوط من هذه الأشواط. يجب أيضاً تحديد كيفية الحصول على المياه الساخنة، هل سيتم تشغيل سخان خاص بالغسالة يتم تشغيله بعد شوط سحب المياه مع تحديد درجة سخونة هذه المياه، أم سيتم إدخال المياه الساخنة من خلال مدخل خاص بها يتم التحكم فيه بصمام معين. يجب تحديد لمبات البيان وعددها، فمثلاً سيكون هناك لمبة بيان تبين الغسالة في حالة تشغيلها، ولمبات أخرى تبين الشوط الذي تعمل فيه الغسالة. يجب التواصل مع مصمم الغسالة للاستفسار عن كيفية التعامل مع المотор المستخدم في هذه الغسالة من حيث النوع، وكيفية عكس حركته، والتحكم في سرعته في كل شوط. من المهم أن تجمع في هذه الخطة كل صغيرة وكبيرة توضح العلاقة بين النظام المدمج الذي ستقوم بتصميمه والنظام الذي سيقوم النظام المدمج بالتحكم فيه.

### ٢- وضع المواصفات Specification

بعد مرحلة التحليل السابقة، عليك أن تقوم بوضع أو تحديد مواصفات للنظام المدمج مثل، عدد خطوط الإشارات المدخلة للنظام، وكم من هذه الخطوط ستكون خطوط رقمية، بمعنى أن الإشارات الداخلية عليها ستكون رقمية مثل الإشارات القادمة من مفاتيح معينة، وكم منها ستكون إشارات تماثلية مثل الإشارات القادمة من حساسات مثل حساس الحرارة، وهل هذه الإشارات ستحتاج للتكبير، والترشيح من الضوضاء، وبالطبع ستحتاج إلى محول تماثلي رقمي ADC، فهل ستستخدم هذا المحول كشريحة منفصلة خارج المتحكم، أم ستستخدم متحكم به مثل هذا المحول. كيف ستتحكم في سرعة المotor هل ستستخدم طريقة تعديل عرض النسبة PWM، pulse width modulation، وهل ستحتاج لعد لفات المotor في كل اتجاه.



شكل ٨ مخطط سير لخطوات تصميم النظام المدمج

هل ستحتاج لمقاطعة تفاصيل البرنامج في حالة حدوث أي أشياء طارئة. ولا ننسى هنا مصادر القدرة وأنواعها وكيفية توفيرها. في نهاية هذه الخطوة يجب أن تحدد نوع المتحكم وإمكانياته والشريحة الأخرى التي ستحتاجها لتنفيذ مكبر الإشارة والمرشحات إذا لزم الأمر، أو باختصار يجب أن تحدد مواصفات كل المكونات المادية hardware التي ستحتاجها، وتقوم بتوفيرها.

## ٢- وضع هيكل للنظام System Architecture

بعد تحديد المكونات المادية والشريحة الإلكترونية يجب أن تحدد كيفية وضعها على اللوحة الإلكترونية وكيفية ترتيبها عليها بحيث يتم اختصار المسارات الواسعة بين الشريحة بعضها البعض، وعلى ضوء ذلك يجب أن تحدد مساحة اللوحة المستخدمة. يجب أن تقرر أيضاً في كيفية طباعة الدائرة وهل ستقوم أنت بهذه المهمة أم ستقوم برسم مخطط للدائرة بالكامل وستعين بآخر يقوم بعملية الطباعة.

## ٣- تصميم المكونات المادية والبرمجية Hardware and software design

تبدأ هذه الخطوة عادة باستخدام أحد برامج المحاكاة مثل برنامج Protues أو أي برنامج آخر تفضله أنت، حيث تقوم بناء النظام بالكامل على المحاكى في أكثر من خطوة ويحسن أن تبدأ بالأجزاء التي خارج المتحكم مثل الحساسات، ومكبرات الإشارة والمرشحات إن وجدت، على أن تبدأ باختبار هذه الأجزاء وحدها أولاً. بعد ذلك تقوم بتوصيل المتحكم مع هذه الدوائر. وهنا تنتقل إلى خطوة تصميم البرنامج الذى سيتم وضعه أو حرقه كما يقال عادة على المتحكم لتشغيل النظام بالكامل. هنا ستنقل إلى برنامج محاكاة البرامج مثل برنامج Atmel Studio الذى يمكنك من خلاله كتابة البرنامج بلغة C وتنفيذها والتخلص من أي أخطاء تظهر فيه، ويمكنك التأكد من ذلك بطرق تصحيح البرنامج المختلفة debugging والتي منها على سبيل المثال تنفيذ البرنامج خطوة بخطوة وتتبع نتائج هذه الخطوات.

هناك قاعدة معروفة ومفيدة في تصميم مثل هذه المشاريع وبالذات المعقدة منها وهي أن **يتم التصميم من أعلى لأسفل**، بينما يتم **بناء المشروع من أسفل لأعلى**. وهذا يعني أنه في مرحلة التصميم نبدأ بتقسيم النظام ككل إلى أنظمة جانبية، ثم نقسم هذه الأنظمة الجانبية إلى أنظمة جانبية أصغر، وهكذا نستمر في عملية التقسيم إلى أن يصل إلى أصغر نظام جانبي يمكنه تنفيذ مهمة واحدة بسيطة. أما في مرحلة تنفيذ المشروع أو بناؤه فإننا نبدأ من هذه النظم الجانبية الصغيرة ثم الأكبر فالأخير إلى أن يتم الانتهاء من النظام بالكامل.

#### **٤-تنفيذ الدائرة والبرمجيات Hardware and software implementation**

هذه الخطوة متداخلة مع الخطوة السابقة حيث يتمأخذ البرنامج بعد التأكد من صحته ووضعه على شريحة المتحكم في برنامج المحاكاة Protues كما أشرنا ويتم اختبار الدائرة بالكامل على برنامج المحاكاة أيضا.

#### **٥-اختبار الدائرة والبرنامج Hardware and software testing**

من المهم جدا في هذه الخطوة اختبار الدائرة وهي على برنامج المحاكاة بالكامل وبكل حالات الاختبار الممكنة بما في ذلك اختبار سلوك النظام في حالة حدوث أي طارئ. لاحظ أن هذه الخطوة متداخلة مع الخطوتين السابقتين.

#### **٦-تجميع النظام System integration**

في هذه الخطوة يتمأخذ آخر نسخة من البرنامج بعد مراحل الاختبار المختلفة وحرقها فعليا على شريحة المتحكم بعد تركيبها في الدائرة على اللوحة الإلكترونية باستخدام أحد البرامج الخاصة بذلك ومنها برنامج Atmel studio وبذلك يصبح النظام بالكامل جاهز للاختبار. يتم إجراء جميع الاختبارات الممكنة على النظام وبالتالي ستظهر هنا أخطاء غير متوقعة وبالذات نتيجة الضوضاء. يتم التخلص من هذه الأخطاء بالعودة إلى مراحل التصميم المختلفة السابقة وإجراء التعديلات المطلوبة. كل هذه المحاولات تتم اللوحة الإلكترونية بالنظام المدمج لم يتم تركيبها فعليا في النظام الأساسي (الغسالة)، وإنما يتم الاختبار من خلال إشارات كهربائية تحاكي الواقع.

#### **٧-التحقق من صحة النظام**

هذه هي الخطوة الأخيرة حيث يتم تركيب اللوحة الإلكترونية بالنظام المدمج في مكانها المعد لتشغيل الغسالة الفعلية، ويتم إجراء تشغيل فعلى للغسالة ودراسة أداء النظام تحت جميع الظروف الممكنة. هنا أيضا ربما يتطلب الأمر العودة إلى الخطوات السابقة وإجراء بعض التعديلات إما في الدوائر أو البرنامج والعودة للاختبار مرة ثانية.

نلاحظ من هذا المخطط أن الخطوات ٤ و ٥ و ٦ مكررة في مسارات متوازيين، أحدهما مسار خاص بالمكونات المادية والآخر خاص بالبرمجة، وأنت كمستخدم لك الحرية في أن تبدأ بأي منها أولا، أو تنفذ جزء من أحد المسارات وتنتقل إلى المسار الآخر وهكذا إلى أن تنتهي من هذين المسارين، وهذه تعتبر خاصية في كل النظم القابلة للبرمجة. من الممكن التحكم في مقدار البرمجة بالنسبة للمكونات المادية. فمثلا بعض المهام يمكن تنفيذها باستخدام مكونات مادية وفي هذه الحالة نستفيد من سرعة التنفيذ بالطبع، أو تنفيذ هذه المهام من خلال البرمجة، وأقرب مثال على ذلك هو المحوّل التماثلي الرقمي ADC. هذا المحوّل يمكن تنفيذه كمكون مادي من خلال شريحة منفصلة، أو نستخدم متحكم يكون

فيه هذا المحوّل. في حالة المواقف التي يكون فيها خوف من طول زمن تنفيذ البرنامج فلا يستطيع النظام العمل في الزمن الحقيقي يفضل استخدام محول ADC خارج المتحكم، وأما إذا كان التنفيذ في الزمن الحقيقي محقق بسهولة وغير حرج، ففي هذه الحالة يكون من الأفضل استخدام المتحكم الداخلي من خلال البرمجة.

## ٨- كتابة وثائق النظام

في كثير من الأحيان يفرح البعض بانتهاء النظام وتركبيه وينسى أن يقوم بتوثيق أو كتابة كل خطوات النظام على الرغم من خطورة هذه الخطوة. إن النظام بالتأكيد سيحتاج بعد فترة للصيانة أو التطوير وستحتاج للرجوع للتفاصيل التي تمت في كتابة البرامج أو في الدوائر الإلكترونية. في هذه الحالة إذا لم يكن لديك توثيق جيد للنظام، فإن موضوع صيانة أو تطويره لن يكون سهلاً على الإطلاق. لذلك يجب العناية بهذه الخطوة جيداً بحيث يكون هناك رسمًا وشرحًا لكل دوائر النظام، مع البيانات الخاصة بكل المكونات المستخدمة، مع مخطط توضيحي لخطوات تنفيذ وتطوير النظام. كذلك البرامج يجب أن تشتمل على توثيق كامل لكل برنامج، وأيضاً التوثيق الكامل لكل أمر إذا أمكن ذلك مع خريطة سير للبرنامج.

في الكثير من الأحيان عند تطويرك لنظام آخر قد تحتاج إلى أجزاء من هذا النظام سواء أجزاء من البرامج أو أجزاء من الدوائر، في هذه الحالة سيكون التوثيق الجيد للنظام مفيداً جداً.

## ٧-١ العوامل المؤثرة في اختيار المعالج أو المتحكم

عند تنفيذ النظم المدمجة لابد أنك ستقف عند اختيار المعالج أو المتحكم المناسب للتعامل مع المشكلة التي تقوم بحلها. من العوامل التي تؤثر في اختيارك للمعالج أو المتحكم ما يلى:

### ١- عدد خطوط إدخال وإخراج البيانات

بعد الانتهاء من مرحلة تصميم النظام لابد أنك تكون قد قررت كم عدد خطوط إدخال وإخراج البيانات. هذا العدد مهم جداً في تحديد اختيارك هل ستلجأ إلى استخدام معالج أم متحكم. إذا كان عدد هذه الخطوط في حدود الثلاثين خط، فإنه بالتأكيد ستتجه أحد المتحكمات الذي يفي لك بهذا الغرض، فهناك مثلاً المتحكم Armega103 الذي يعطيك حتى ٣٢ خط إدخال وإخراج للبيانات، كما أن هناك المتحكم Atiny22 الذي له ٨ أرجل منها ٦ أرجل للإدخال والإخراج للبيانات، وهناك العديد من المتحكمات المختلفة الأخرى التي تعطي خطوط إدخال وإخراج في هذه الحدود، وكلها من متحكمات AVR من شركة أتميل. أما إذا كان عدد الخطوط أكثر من ذلك كما في بعض التطبيقات

الصناعية التي يكون بها العديد من لمبات البيان والكثير من الحساسات المختلفة، ففي هذه الحالة لن يكون هناك فرار من استخدام أحد المعالجات. ويجب ألا يكون هناك أي خوف من ذلك على الإطلاق فهناك الكثير من المعالجات البسيطة والسهلة في التعامل مثل المعالج Z80 الذي يمكن به استخدام حتى ٢٥٦ خط إخراج للبيانات، و ٢٥٦ خط آخر لإدخال البيانات، وهذا العدد بالتأكيد سيكون أكثر من كاف مع العديد من التطبيقات. فما بالك بالمعالج intel8086 الذي يمكنك به الوصول إلى عدد خطوط إخراج وإدخال تصل إلى ٦٥٥٣٦ خط. الخلاصة من ذلك هي أنه يجب استخدام المتحكمات طالما أن ذلك ممكن لسهولة التعامل معها وبرمجتها عن المعالجات التي تحتاج للكثير من دوائر المواجهة في التعامل معها.

## ٢- هل البيانات التي ستتعامل معها متوازية أم متتالية

يجب أن يكون المتحكم الذي ستستخدمه مناسب أو به إمكانية للتعامل مع نوع البيانات المقترحة. تقريبا كل المتحكمات تعامل في الأصل من خلال البيانات المتوازية، مما يعني نقل البيانات على مسارات متوازية من ٨ أو ١٦ أو أكثر من ذلك من الباتات. في بعض التطبيقات ستحتاج لنقل البيانات على خط واحد (وليكن خط تليفون مثلا) إلى طرف آخر، لذلك لابد من اختيار المتحكم الذي به هذه الإمكانيات. ربما يحتاج التطبيق الذي تعامل معه إلى بيانات معدلة النبضة pulse width modulated, PWM وإنما ينصح بـ ATmega328 الذي ستركز عليه في هذا الإمكانية. بالنسبة فإن معظم متحكمات AVR والتي منها المتحكم Atmega328 الذي سنركز عليه في هذا الكتاب يوفر هذه الإمكانيات وأكثر.

## ٣- كمية الذاكرة المطلوبة

توجد المتحكمات مزودة بكميات مختلفة من ذاكرة ال RAM وال ROM، وبالتالي سيحدد سعر المتحكم على حسب هذه الكميات. لذلك لابد أن يكون هناك اختيار حكيم لهذه الكميات. ذاكرة القراءة فقط هي التي سيتم تسجيل البرنامج عليها ليتم تنفيذه بمجرد توصيل القدرة للنظام، ولذلك لابد أن يكون لديك تصور عن حجم برنامجك وكمية ذاكرة القراءة فقط (غير المتطابقة) المطلوبة له. قد يكون التطبيق الذي تعامل معه يحتاج للتسجيل المستديم لبعض الثوابت أو جداول البحث look up tables، لذلك يجب عمل حساب ذلك في كمية ال ROM المطلوبة. السؤال الثاني الذي يجب أن تجيب عليه هو هل ستحتاج إلى كمية من الذاكرة العشوائية RAM أم لا، وما مقدارها؟ عموما إذا كانت برمجتك ستستخدم برامج فرعية subroutines أو مقاطعة Interrupt فإنه لابد من توفر كمية من الذاكرة العشوائية لاستخدامها كمكشدة stack يوضع بها عناوين العودة من البرامج الفرعية أو برامج خدمة المقاطعة

Interruption Service Routines, ISRs. بما يكون التطبيق الذى تعامل معه يحتاج لبعض الحسابات البينية التى تحتاج لكمية من ال RAM، لذلك يجب أن تعمل حساب ذلك. عموماً يجب أن تختار متحكم بكمية RAM و ROM مناسبتين للتطبيق الذى تعامل معه ويجب أن تختار هذه الكميات بوفرة معقولة بحيث لا تكون مضطراً لتعديل المتحكم إذا احتجت زيادة في أي كمية من هذه الذاكرة فيما بعد أثناء تطوير النظام. عموماً إذا كان التطبيق المقترن يستخدم ٨٠٪ من الذاكرة الملحقة بالتحكم فهذا المتحكم غير مناسب ويجب أن تختار متحكم آخر بإمكانيات أكبر.

#### ٤- عدد خطوط المقاطعة

هل ستستخدم المقاطعة في برنامجك أم لا؟ إذا كنت تستخدمها، فيجب أن يكون المتحكم الذي ستختاره به هذه الإمكانيات. في العادة تكون الأجهزة الملحقة بالتحكم بما إمكانيات المقاطعة، فمثلاً تحول التماثلي الرقمي ADC به إمكانية أن يقاطع البرنامج الأساسي الذي يتم تنفيذه ليعطي القيمة الرقمية بعد الانتهاء من التحويل. أيضاً فإن المؤقتات الملحقة بالتحكم يمكنها أيضاً مقاطعة المتحكم عند انتهاء زمن التوقيت. يجب أن تقرر هل ستستفيد من هذه الميزات أم لا.

#### ٥- طريقة كتابة البرنامج

أثناء تطويرك للبرنامج، هل ستستخدم لغة التجميع Assembly language، أم ستستخدم أحد اللغات العالية المستوى مثل لغة C أو لغة الباسيك، وقد أصبح هناك إصدارات من هذه اللغات متاحة الآن للاستخدام في تطوير برامج المتحكمات. بالطبع فإن استخدام لغات المستوى العالى يكون أسهل عن استخدام لغة التجميع ونحن ننصح بذلك.

#### ٦- المعالجة في الزمن الحقيقى

هناك الكثير من التطبيقات التي يكون التعامل في الزمن الحقيقى فيها غاية في الأهمية نتيجة التغير السريع في الإشارة التي يتم التحكم فيها أو قراءتها. فمثلاً لو أنك تعامل مع إشارة صوت التي من الممكن أن يصل تردداتها إلى ٢٠ كيلوهرتز، فإنه لابد من أن يكون معدل قراءة العينات منها ، ٤ كيلوهرتز على الأقل. إن ذلك يعني أن الزمن بين كل عينة من هذا الصوت وبالتالي لها سيكون  $10 \times 40 / 1$  ثانية وهو ما يساوى ٢٥ ميكروثانية. إن ذلك يعني أنه في البرنامج الذى ستكتبه يجب أن يكون زمن قراءة كل عينة، زائد زمن معالجة هذه العينة (هذه المعالجة قد تكون ضرب

هذه العينة في ثابت، أو مقارنتها بثابت آخر، أو حتى تخزينها في الذاكرة)، زائد زمن إخراج أي قرار يتوقف على هذه العينة، مجموع هذه الأزمنة يجب أن يكون أقل من الزمن الفاصل بين عينتين وهو ٢٥ ميكروثانية، وبمعامل أمان كافٍ. إذا لم يتحقق ذلك فإنه سيكون هناك فقد للكثير من عينات الصوت وستفقد الكثير من خواصه. على ضوء هذه الأزمنة سيتحدد زمن تنفيذ أوامر المتحكم وسيتحدد تردد نبضات التزامن الخاص به.

## ملخص الفصل

بذلك تكون قد انتهينا من هذا الفصل الذي تم فيه استعراض معنى النظم المدمجة، وخصائصها، والطريقة المثلث المقترحة لتدريس مثل هذا المقرر. ولقد تم أيضا تقديم نظرة تاريخية سريعة عن تطور الإلكترونيات منذ ظهور الترانزistor وحتى آخر المعالجات، ودور قانون مور في ذلك. لقد تم أيضا استعراض خطوات تنفيذ أي نظام مدمج. العوامل التي يجب أخذها في الاعتبار عند اختيار أي معالج أو متحكم ليتمثل قلب النظام المدمج تم استعراضها أيضا. كان هناك أيضا تعريف بعض المصطلحات مثل وحدة المعالجة المركزية CPU، المعالج (الميكروبروسيسور)، والميكروكمبيوتر، والمتحكم (الميكروكونتroller). وحدة المعالجة المركزية هي قلب أي شريحة حاسب كما ذكرنا، ولا بد أن يكون المتعامل مع أي منها على دراية بتفاصيل هذه الوحدة (وحدة المعالجة المركزية CPU)، وهذا هو موضوع الفصل التالي.

## الفصل ٢

### وحدة المعالجة المركزية

### Central Processing Unit, CPU

**العناوين المضيئة في هذا الفصل:**

- ١- المهام الأساسية المطلوبة من وحدة المعالجة المركزية.
- ٢- نظرة عامة على أطراف وحدة المعالجة المركزية
- ٣- دورة الكتابة والقراءة من الذاكرة
- ٤- توصيل المسارات بين المعالج والذاكرة
- ٥- التركيب الداخلي لوحدة المعالجة المركزية
- ٦- أنواع الذاكرة

## ١-٢ مقدمة

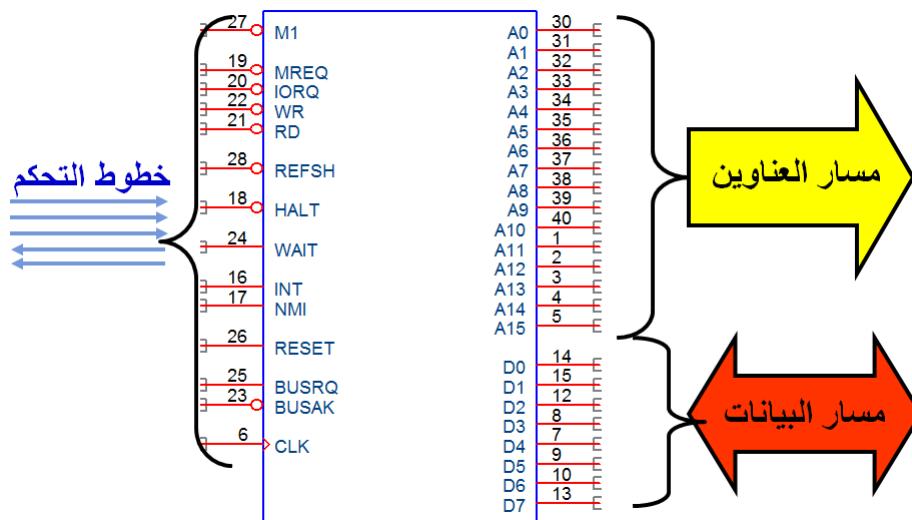
**سيتم** في هذا الفصل عرض المهام الأساسية المطلوبة من أي وحدة معالجة مركزية CPU (معالج) بصفة عامة وعلى ضوء هذه المهام سنعرض الوظائف الأساسية لأطراف ومكونات أي شريحة معالج. لقد رأينا في الفصل السابق (المقدمة) أن وظيفة المعالج الأساسية هي إحضار الأوامر من الذاكرة وتنفيذها الواحد بعد الآخر, ولذلك فإن أطرافه وتركيبيه الداخلي يجب أن يناسب هذه المهمة.

## ٢-٢ المهام الأساسية المطلوبة من المعالج

١. يجب أن يكون المعالج (وحدة المعالجة المركزية CPU) قادرًا على إحضار معلومات من الذاكرة (هذه المعلومات قد تكون بيانات يحتاجها في عملية تنفيذ الأوامر أو قد تكون الأوامر نفسها).
  ٢. يجب أن يحتوى المعالج على مكان مناسب بداخله لحفظ هذه المعلومات التي أحضرها لحين الحاجة إليها أو تنفيذها إذا كانت من الأوامر.
  ٣. لابد أن يكون هناك أكثر من مكان بداخله بحيث يمكن نقل المعلومات فيما بين هذه الأماكن حيث تحتاج بعض الأوامر لذلك عند تنفيذها.
  ٤. يجب أن تكون لديه الوسائل المناسبة لإدخال معلومات من بوابات إدخال حتى يتسعى لنا قراءة لوحة مفاتيح أو إدخال درجة حرارة مثلاً تمهيداً لمعالجتها رقمياً.
  ٥. يجب أن تكون لديه المقدرة على إجراء بعض العمليات الحسابية والمنطقية على البيانات التي أحضرها. العمليات الحسابية الأساسية هي الجمع والطرح والعمليات المنطقية الأساسية مثل AND و OR و NOT.
  ٦. المقدرة على إرسال بيانات إلى الذاكرة وتسجيلها فيها من المهام الأساسية للمعالج.
  ٧. المقدرة على إرسال بيانات إلى وحدات إخراج من خلال بوابات إخراج حتى يتسعى لنا قراءة هذه المعلومات على شاشة أو إخراج بيانات تحكم بها في سرعة موتور مثلاً.
- كانت هذه هي المهام الأساسية للمعالج والتي يجب أن يتحققها تركيبة المعالج وأطرافه ومجموعة أوامره كما سنرى. سنبدأ فيما يلى بإلقاء نظرة على أطراف شريحة المعالج من الخارج ووظيفة كل طرف من أطرافه التي تسهل من مهمة المعالج الأساسية التي هي إحضار الأوامر من الذاكرة وتنفيذها. يأتي بعد ذلك الحديث عن التركيب الداخلى لشريحة المعالج.

## ٣-٣ نظرة عامة على أطراف شريحة المعالج

ما سبق نلاحظ أن التعامل مع الذاكرة والأجهزة المحيطة من بوابات إدخال وإخراج هو من المهام الأساسية لوحدة المعالجة المركزية. لذلك لابد من وسيلة يتم عليها تحديد العنوان الذي سيتم التعامل معه في الذاكرة وهو ما يسمى بمسار العناوين address bus، ومسار يتم عليه نقل البيانات التي سيتم إرسالها أو استقبالها من الذاكرة، وهو ما يسمى بمسار البيانات data bus، ثم بعد ذلك لابد من وجود خطوط تحكم تحدد للذاكرة أو للأجهزة الخارجية نوع هذا التعامل، هل هو بعرض القراءة أم بعرض الكتابة بالإضافة إلى الكثير من أغراض التحكم الأخرى مثل المقاطة interrupt وإعادة الوضع reset، وتحديد نوع التعامل من حيث تعامل مع الذاكرة أم مع بوابات إدخال أم مع بوابات إخراج للبيانات، وهكذا. أى أن أطراف وحدة المعالجة المركزية CPU أو المعالج لن تخرج عن كونها أطراف لمسار عناوين، أو مسار بيانات، أو خطوط تحكم. شكل ١-٢ يبين أطراف الشريحة Z80 وهي من أشهر المعالجات ذات ٨ بت والتي تعتبر مثالاً جيداً لوحدة المعالجة المركزية التي نريد دراستها، وكل المعالجات الأخرى لن تبتعد كثيراً عن هذا المثال.



شكل ١-٢ نظرة على أطراف وحدة المعالجة المركزية CPU

كل شرائح المعالجات ٨ بت التي أنتجت في ذلك الوقت كانت تتكون من ٤٠ طرف وشكل ١-٢ كما ذكرنا يبين أطراف المعالج Z80 كمثال على ذلك. نلاحظ من هذا الشكل أن هذه الأطراف مقسمة إلى ثلاث مجموعات. مجموعة الخطوط الأولى هي مسار العناوين الذي يتكون من ١٦ خط أو ١٦ طرف، والمجموعة الثانية هي مسار البيانات الذي

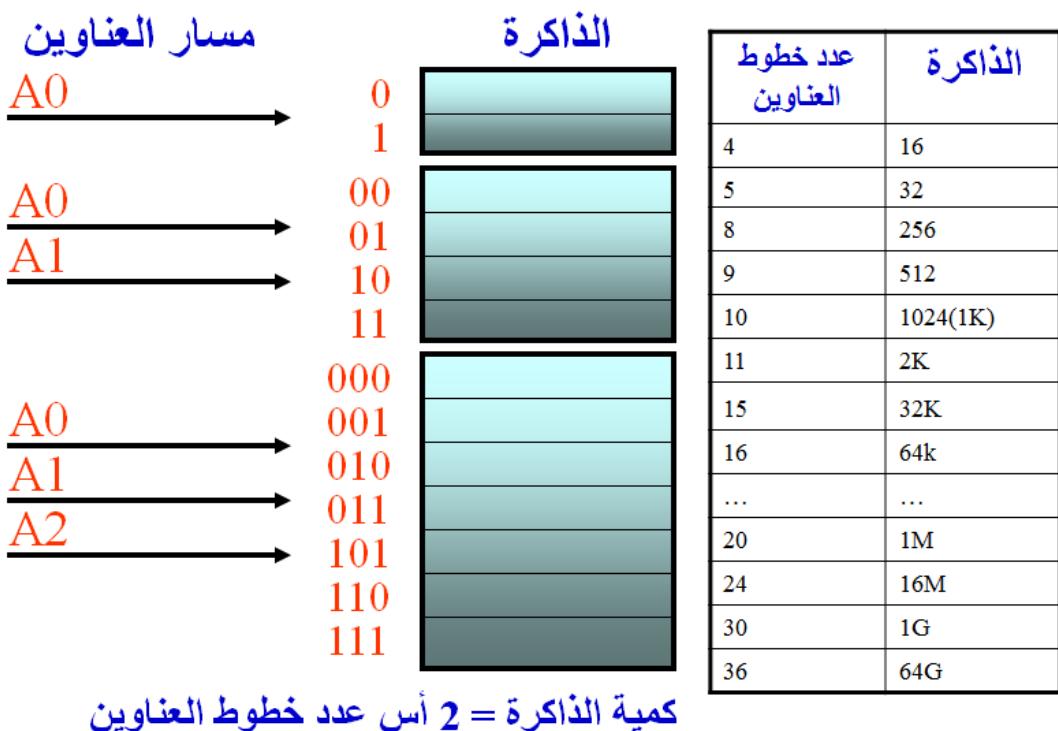
يتكون من ٨ خطوط، ثم في النهاية هناك ٤ خطأ أو طرفا تمثل خطوط التحكم، ولا ننسى وجود خطين للقدرة (VCC والأرضي) وهما غير موضحين في هذا الرسم. فيما يلى سنتكلم بشيء من التفصيل عن كل واحدة من هذه الجاميع لنفهم الوظيفة الأساسية لكل منها، ومواصفاتها، وتأثير تقليل أو زيادة عدد خطوط هذه الجاميع على أداء المعالج.

## مسار العناوين

وظيفة هذا المسار هي نقل إشارة العناوين في صورتها الرقمية (وحاید وأصفار) من المعالج إلى الأجهزة المحيطة به (الذاكرة وبابات إخراج وإدخال البيانات). أى أن كل خط سيحمل أحد بثات هذه الإشارة. فإذا كان العنوان الذى يخرج منه المعالج مكون من ١٦ بت فإن مسار العناوين سيكون ١٦ خط بحيث تخرج كل بت من بثات هذا العنوان على خط منفصل من خطوط المسار. لاحظ أن مصدر العناوين دائمًا هو المعالج، أى أن إشارة العناوين تكون دائمًا خارجة من المعالج إلى الأجهزة المحيطة، ولا يمكن أن تكون عكس ذلك لأن المعالج دائمًا يكون هو المسئول عن تحديد وإخراج العناوين بناء على أوامر البرامج التي ينفذها. تذكر أننا نعني بالمعالج أنه وحدة المعالجة المركزية CPU، مثل الموجودة في قلب أي متحكم من المتحكمات.

لكى نفهم تأثير عدد خطوط هذا المسار على أداء المعالج تعالى نفترض أن المعالج كان يخرج عناوين من بت واحدة فقط. في هذه الحالة سيكون المعالج قادرًا على التعامل مع مكانيين مختلفين فقط، المكان الأول سيكون عنوانه ٠ والمكان الثاني سيكون عنوانه ١. سنفترض أن المعالج سيتعامل مبدئياً مع ذاكرة، في هذه الحالة سيتعامل المعالج مع ذاكرة من ٢ بآية فقط البايت الأولى عنوانها ٠ والثانية عنوانها ١ كما ذكرنا. لو فرضنا زيادة العنوان الذي يخرج منه المعالج إلى ٢ بت، في هذه الحالة ستكون الذاكرة التي يتعامل معها المعالج مكونة من ٤ بآية، بحيث يكون عنوان البايت الأولى هو ٠٠، والثانية عنوانها ٠١، والثالثة عنوانها ١٠، والرابعة سيكون عنوانها ١١. وهكذا لو زاد عدد بثات العنوان وبالتالي عدد خطوط العناوين إلى ثلاثة بدلاً من اثنين فإن الذاكرة التي يتعامل معها المعالج ستكون من ٨ بآية عناوينها كالتالي: ٠٠٠ و ٠٠١ و ٠٠١٠ و ٠٠١١ و ٠١٠ و ٠١١ و ١١٠ و ١١١. نلاحظ من ذلك أنه بزيادة عدد خطوط العناوين بمقدار خط واحد فإن العناوين التي يمكن للمعالج أن يتعامل معها ستتضاعف. العلاقة بين كمية الذاكرة التي يمكن للمعالج أن يتعامل معها وعدد خطوط العناوين هي أن كمية الذاكرة تساوى ٢ أَسْ عدد خطوط العناوين. فإذا كان عدد خطوط العناوين يساوى ٨ خطوط فإن كمية الذاكرة ستتساوى ٢ أَسْ ٨ وهو ما يساوى ٢٥٦ بآيت، وإذا كان عدد خطوط العناوين ١٦ خط، فإن الذاكرة التي يتعامل معها المعالج ستكون ٢ أَسْ ١٦ وهو ما يساوى ٦٥٥٣٦، أى ما يساوى ٦٤ كيلوبايت، وهذا هو الوضع في كل المعالجات ٨ بت كما في شكل ١-٢. شكل ٢-٢

يبين رسمًا توضيحيًا للعلاقة بين عدد خطوط العنوان وكمية الذاكرة بالبايت التي يمكن للمعالج أن يتعامل معها كما يشتمل الشكل على جدول يبين هذه العلاقة حتى عدد خطوط عنوانين يساوى ٣٦ خطاً (وهو الحال في المعالج بتقنية ٤) حيث تكون الذاكرة عندها تساوى ٢ أس ٣٦ وهو ما يساوى ٦٤ جيجابايت.



شكل ٢-٢ رسم تخطيطي وجدول يبين العلاقة بين كمية الذاكرة وخطوط العنوان الخارج من المعالج

دائماً يوصف مسار العنوانين بأنه أحادى الاتجاه، بمعنى أن الإشارة على مسار العنوانين تكون في اتجاه واحد فقط من المعالج إلى الأجهزة الخارجية، ولا يمكن أن تكون بالعكس.

الذاكرة التي تتحدد بعدد خطوط مسار العنوانين هي ما يسمى بالذاكرة الأساسية. الذاكرة الأساسية هي الذاكرة التي لابد من وضع أى برنامج فيها لكي يمكن للمعالج أن ينفذه. وهذه الذاكرة تختلف عن الذاكرة الثانوية مثل الأسطوانة الصلبة hard disk والاسطوانات المدمجة CD وغيرها من الذاكرات الثانوية أو الجانبية والتي لا علاقة لها بعدد خطوط مسار العنوانين. هذه الذاكرة الأساسية جزء كبير منها أو معظمها هو ما يعرف باسم ال RAM الملحق بمحاسب كل منها، وجزء منها يكون ذاكرة غير متقطعة nonvolatile أو ذاكرة قراءة فقط لحفظ ثوابت الحاسوب الضرورية لتشغيله.

## مسار البيانات

تننتقل البيانات بين المعالج والأجهزة المحيطة والعكس على خطوط مسار البيانات، لذلك فإن مسار البيانات يكون ثنائياً الاتجاه، يحمل الإشارات من المعالج إلى الأجهزة المحيطة في حالة التسجيل في هذه الأجهزة أو من الأجهزة المحيطة إلى الذاكرة في حالة القراءة منها. لاحظ أن المعالج من المستحيل أن يسجل في الذاكرة مثلاً ويقرأ منها في نفس الوقت، ولكن ذلك يتم في أوقات متتابعة. شكل ١-٢ يوضح مسار البيانات بسمه ذو رأسين دلالة على ثنائية الاتجاه. مسار البيانات في كل المعالجات ٨ بت يتكون من ٨ خطوط، وتعبير ٨ بت في هذه المعالجات يدل على عدد خطوط مسار البيانات. فالمعالجات ١٦ بت تعني أن لها مسار بيانات ١٦ خط، والمعالجات ٣٢ بت تعني أن مسار البيانات بها ٣٢ خط، وهكذا، فالمعالج بنتيجة ٤ مثلاً له مسار بيانات ٦٤ طرف، لذلك نقول عنه أنه معالج ٦٤ بت.

من المعلوم أن وحدة الذاكرة هي البایت (٨ بت)، لذلك فإنه عندما يكون عدد خطوط مسار البيانات يساوى ٨ خطوط فإن ذلك يعني أنه سيتم نقل بایت كاملة في كل تعامل سواء في حالة القراءة أو الكتابة. معنى ذلك أن المعالج بنتيجة ٤ الذي له مسار بيانات مكون من ٦٤ خط يستطيع نقل ٨ بایت في كل تعامل سواء في حالة القراءة أو الكتابة. إن ذلك يعني أن عدد خطوط مسار البيانات يكون تأثيره على سرعة نقل البيانات بين المعالج والأجهزة المحيطة. نؤكد هنا أن المؤثر الأساسي في أداء المعالج من حيث سرعة تنفيذ الأوامر هي نبضات الساعة التي يعمل عندها المعالج والتي سنتكلم عنها بعد قليل.

## خطوط التحكم

خطوط التحكم هي خطوط منفصلة وكل خط تكون له مهمة معينة، ولذلك نطلق عليها خطوط وليس مسار لأن المسار تكون خطوطه كلها لها نفس المهام والإشارة عليها كلها تكون في نفس الاتجاه. من هذه الخطوط خط للقراءة RD وهذا الخط تكون عليه إشارة يخرجها المعالج ليخبر الأجهزة الخارجية عن الغرض من التعامل وهو القراءة منها. بالمثل هناك خط الكتابة WR وهو إشارة خارجة أيضاً من المعالج ليخبر الأجهزة الخارجية بأن الغرض من التعامل هو الكتابة فيها. هناك أيضاً الخط INTR الذي عند تنشيطه من خارج المعالج فإن المعالج يترك البرنامج الذي ينفذه ويقفز إلى برنامج جانبي يسمى برنامج خدمة المقاطعة يقوم بتنفيذها، وبعد الانتهاء منه يعود إلى البرنامج الذي خرج منه. هناك الكثير من مثل هذه الخطوط بنفس النمط، وعدد هذه الخطوط مختلف من معالج إلى آخر. ونحن هنا لن نشرح وظائف كل واحد من هذه الخطوط بنفس النمط، وعدد هذه الخطوط مختلف من معالج إلى آخر. ونحن هنا لن حاجة للتعامل المباشر مع هذه الخطوط، وما يهمنا هنا باختصار أن خطوط التحكم يكون كل خط فيها له وظيفة

محددة وتكون الإشارة عليه إما خارجة من المعالج تخبر الأجهزة الخارجية عن الغرض من التعامل أو داخلة إليه من الأجهزة المحيطة تخبره بعمل شيء.

Atmega 328	
PC6	1
PD0	2
PD1	3
PD2	4
PD3	5
PD4	6
VCC	7
GND	8
PB6	9
PB7	10
PD5	11
PD6	12
PD7	13
PB0	14
PC5	28
PC4	27
PC3	26
PC2	25
PC1	24
PC0	23
GND	22
AREF	21
AVCC	20
PB5	19
PB4	18
PB3	17
PB2	16
PB1	15

شكل ٣-٢ أطراف المتحكم

في نهاية هذا الجزء فإن النظرة الخارجية على أطراف المعالج (CPU) نجد أنها لن تخرج عن كونها إما أطراف في مسار العناوين، أو أطراف في مسار البيانات، أو أطراف تحكم، أو طرف قدرة كما في شكل ١-٢ . بهذه المناسبة فإن شكل ٣-٢ يبين أطراف شريحة المتحكم كمثال على واحد من هذه المتحكمات. نلاحظ من هذا الشكل أن أطراف المتحكم منقسمة إلى ثلاث بوابات، البوابة B، والبوابة C، والبوابة D وكلها بوابات يمكن برمجتها لإدخال البيانات إلى المتحكم أو إخراج البيانات من المتحكم. عدد خطوط هذه البوابات هي ٢٣ خطًا بالإضافة إلى خمس خطوط قدرة فيصبح المجموع هو ٢٨ خط هي كل خطوط الشريحة. نلاحظ أنه ليس من بين هذه الخطوط مسار للعناوين أو للبيانات أو للتحكم. والسبب في ذلك أن وحدة المعالجة المركزية،

والذاكرة، والكثير من الأجهزة المحيطة توجد كلها بداخل المتحكم كما سررى بعد ذلك بالتفصيل، ويتم التعامل معها داخلياً من خلال مسارات للعناوين والبيانات والتحكم داخلياً، ولا حاجة لإخراج هذه الإشارات على أطراف خارج الشريحة لأنه لن يتم توصيل لا ذاكرة ولا أي جهاز آخر يحتاج لعنونة خارج حدود الشريحة. إن هذا يعتبر فرقاً جوهرياً بين شريحة المعالج (CPU) وشريحة المتحكم، وهو أن كل أطراف شريحة المعالج تكون أطراف مسارات (عناوين، أو بيانات، أو تحكم)، بينما أطراف المتحكم تكون كلها أطراف إدخال أو إخراج للبيانات، وهذا يتضح تماماً من شكل ٣-٢ و ٤-٢ .

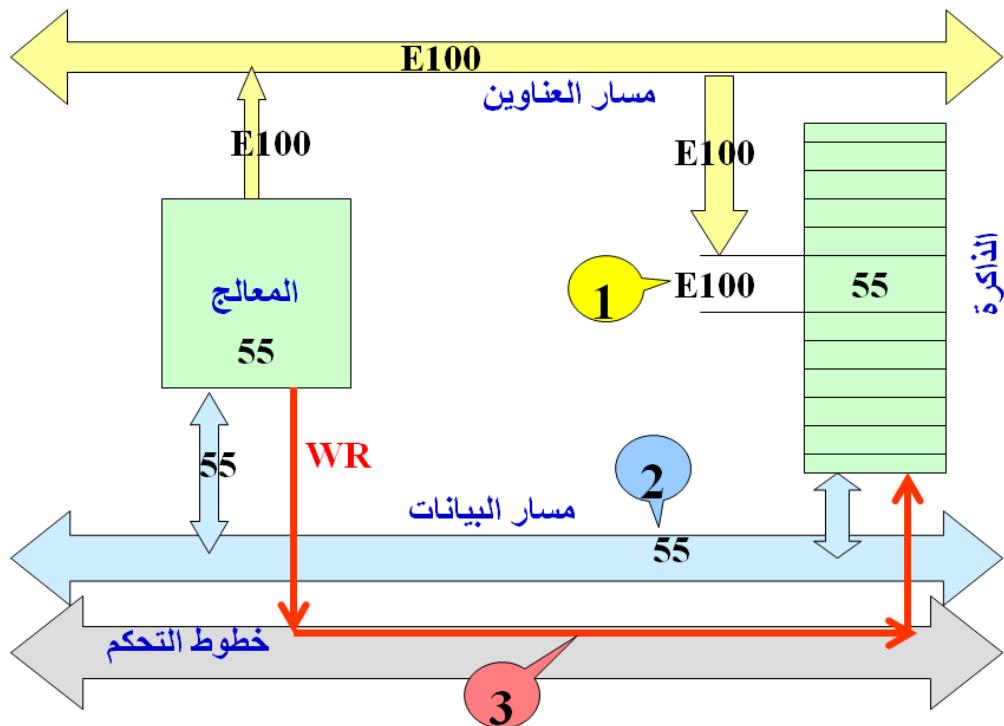
## ٤-٤ دورة القراءة من الذاكرة والكتابة فيها

يمكننا أن نفهم دور كل واحد من المسارات السابقة إذا تبعنا ماذا يحدث أثناء عملية القراءة من الذاكرة أو الكتابة فيها. أي واحدة من هاتين العمليتين تتم في ثلاثة خطوات وكل خطوة تتعلق بأحد المسارات كما في شكل ٤-٢ الذي

يوضح دورة الكتابة أو التسجيل في الذاكرة. في هذا الشكل العنوان E100H هو عنوان البابت المراد التسجيل فيها، والمعالج يحتوى المعلومة 55H، والمعالج يريد تسجيل هذه المعلومة في هذا العنوان. تذكر أن الحرف H يعني أن الرقم الذى قبله مكتوب فى النظام المستعشرى hexadecimal. لاحظ أن العنوان E100 فى النظام المستعشرى يتكون من 16 بت، وبالتالي يخرج على 16 طرفا من أطراف المعالج وهى مسار العناوين. كذلك فإن المعلومة 55 فى النظام المستعشرى تتكون من 8 بت وسيتم نقلها من المعالج على مسار بيانات من 8 أطراف. عملية الكتابة تتم فى الخطوات الثلاث التالية بالترتيب:

- 1 - يضع المعالج العنوان (16 بت) الذى يريد التعامل معه على مسار البيانات (16 طرف).
- 2 - يضع المعالج المعلومة 55H الموجودة بداخله على مسار البيانات.
- 3 - يقوم المعالج بتنشيط خط التحكم WR بإشارة تخرج منه إلى الذاكرة تخبرها أن الغرض من هذا التعامل هو الكتابة أو التسجيل في الذاكرة، لذلك يتم تنشيط هذا الخط WR.

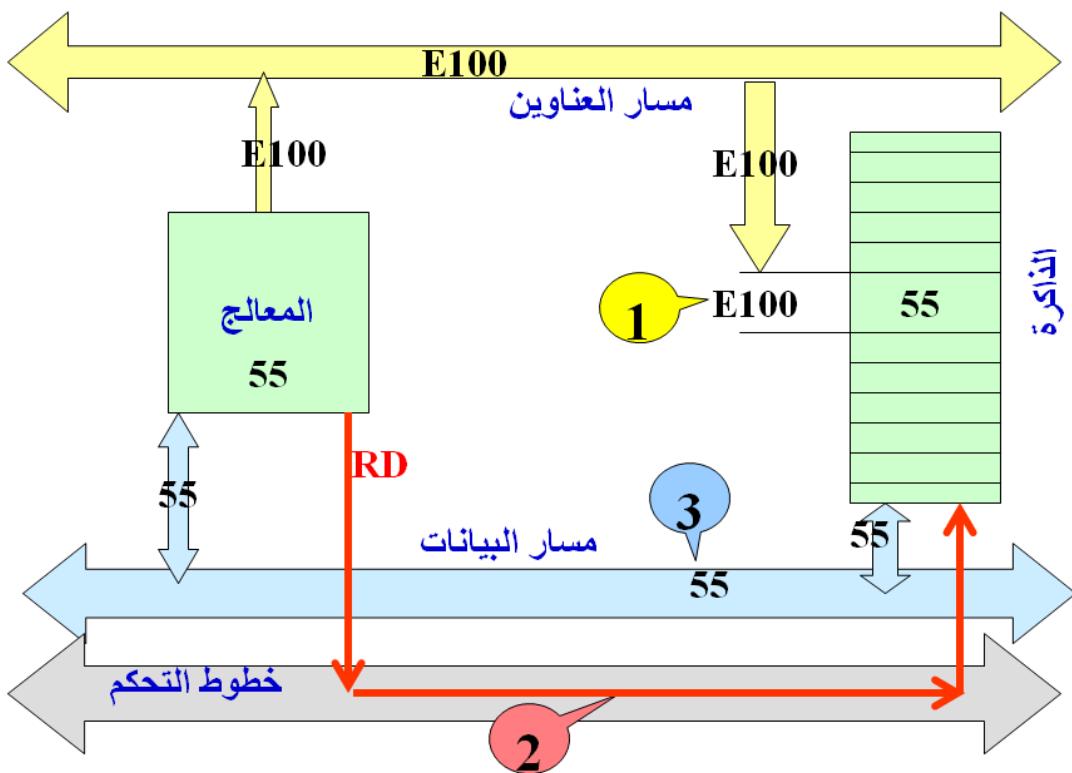
لاحظ أن الخطوات الثلاث متعلقة بالمسارات الثلاثة (العناوين، والبيانات، والتحكم)، وبعد تنفيذ هذه الخطوات يتم نقل نسخة من المعلومة 55H وتسجيلها في العنوان المحدد E100H.



شكل ٤-٢ رسم تخطيطى لعملية التسجيل في الذاكرة

عملية القراءة من الذاكرة وهى عملية عكسيّة لعملية التسجيل في الذاكرة تم أيضًا في ثلاثة خطوات متعلقة بالمسارات الثلاثة (العناوين، والبيانات، والتحكم)، وهى تتم في الخطوات الثلاث التالية بالترتيب كما في شكل ٥، وسنفترض أننا نريد قراءة المعلومة 55H التي كتبناها مسبقاً من نفس العنوان E100H:

- ١- يقوم المعالج بوضع عنوان البايت E100H على مسار العنوانين.
- ٢- هذه المرة يقوم المعالج بتنشيط خط التحكم RD ليخبر الذاكرة بأنه يريد القراءة من هذا العنوان.
- ٣- على الفور تقوم الذاكرة بوضع نسخة من المعلومة الموجودة في هذه البايت وهي 55H على مسار البيانات، حيث يقوم المعالج بالتقاطها من على مسار البيانات وتسجيلها في المكان المناسب بداخله.



شكل ٥-٢ رسم تخطيطي لعملية القراءة من الذاكرة

نلاحظ من هاتين العمليتين أن عملية القراءة ليست مدمرة للمكان الذي تم القراءة منه حيث يتم فقطأخذ نسخة من هذه المعلومة مع عدم تغيير محتويات المكان الذي تم قراءته. بينما عملية الكتابة تعتبر عملية مدمرة حيث أنه يتم مسح أو ضياع أي معلومة في المكان الذي تم الكتابة فيه ويوضع بدلاً منها المعلومة الجديدة. وهذه تعتبر شبه قاعدة

ثابتة في عالم الحاسوبات، حيث أنه عند قراءة محتويات أي مكان في الذاكرة، فإنه يتم أخذ نسخة من محتويات هذا المكان وتبقى محتوياته كما هي. بينما عند الكتابة في أي مكان في الذاكرة، فإن محتويات المكان الأصلية الذي يتم الكتابة فيه تفقد ويوضع بدلاً منها المحتويات الجديدة التي يتم تسجيلها.

## ٥-٢ نظام توصيل المسارات بين المعالج والذاكرة

### Bussing system

عندما يتعامل المعالج مع الذاكرة، يتم بواحدة من الطريقتين التاليتين:

١- طريقة فان نيومان Van Neumann للتوصيل بين المعالج والذاكرة:



شكل ٦-٢ طريقة فان نيومان للتوصيل الذاكرة على المعالج

فان نيومان هو عالم انجليزي وهو أول من اقترح تقسيم المسارات بين المعالج والأجهزة المحيطة إلى ثلات مسارات (العنوانين، والبيانات، والحكم) واقتراح فكرة وضع البرنامج في الذاكرة على أن تقوم وحدة المعالجة المركزية بالنداء على أوامر البرنامج الواحد بعد الآخر حتى يتم الانتهاء من البرنامج. لذلك، فإن هذه الطريقة تتميز بوجود **ذاكرة واحدة تتعامل معها وحدة المعالجة المركزية CPU تحتوي البرامج والبيانات معاً**، كما في شكل ٦-٢. نلاحظ من هذا الشكل وجود ذاكرة واحدة ويت التعامل معها من خلال المسارات الثلاثة. وحدة التعامل مع الذاكرة هي البايت (٨ بت)، والمشكلة أن الكثير من أوامر وحدة المعالجة المركزية تتكون شفراتها من أكثر من بايت واحدة، ولذلك لابد من كتابة كل أمر من هذه الأوامر في أكثر من بايت في الذاكرة، ولذلك فإن مثل هذه الأوامر ستقوم وحدة المعالجة المركزية بإحضارها في أكثر من عملية قراءة من الذاكرة، فالأمر المكتوب مثلاً في ٢ بايت سيتم إحضاره من خلال عملية قراءة (مشوارين) من الذاكرة. لذلك فإن إحضار الأوامر من الذاكرة في هذه الطريقة يكون أبطأ من الطريقة التالية التي سنراها في الجزء التالي. كل

أجهزة الحاسوب العامة (لاب توب وغيرها) تقوم على هذه الطريقة، حيث أنها تكون الأبسط وبالتالي الأرخص في التو�صيل بين المعالج والذاكرة.

## ٢- طريقة هارفارد Harvard للتوصيل بين المعالج والذاكرة:

الإسم هارفارد نسبة لجامعة هارفارد أو من استخدمت هذه الطريقة. في هذه الطريقة تم فصل الذاكرة إلى جزأين، جزء خاص بالبيانات يتم التعامل معه بوحدة البايت العادي (٨ بت) ويتصل بوحدة المعالجة المركزية من خلال مسار بيانات ومسار عناوين خاص بهذا الجزء من الذاكرة. الجزء الثاني هو ذاكرة خاصة بالبرامج فقط، تكون وحدة التعامل معه هي عدد من البتات يساوي أكبر عدد من بباتات أي أمر من الأوامر، فإذا كانت أكبر شفرة للأمر تكون مثلاً من ١٤ بت فإن وحدة التعامل مع هذه الذاكرة ستكون وورد (كلمة) مكونة من ١٤ بت، مما يعني أن مسار البيانات لهذه الذاكرة سيكون ١٤ خط. وإذا كانت أكبر شفرة للأمر مكونة من ١٦ بت، فإن وحدة التعامل مع هذه الذاكرة ستكون ١٦ بت وبالتالي فإن مسار البيانات سيكون ١٦ خط، وهكذا. بالطبع سيكون هناك مسار عناوين آخر خاص بهذه الذاكرة عدد خطوطه يتحدد بكمية هذه الذاكرة. شكل ٧-٢ يبين رسمياً تخطيطياً لهذه الطريقة.



شكل ٧-٢ طريقة هارفارد للتوصيل الذاكرة على المعالج

## ٦-٢ التركيب الداخلى لوحدة المعالجة المركزية

### CPU Architecture

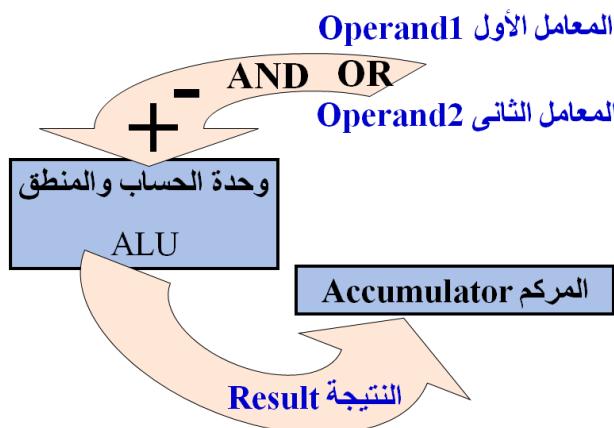
جميع شرائح المعالجات تتربّك من ثلاثة أجزاء رئيسية وهي:

١- وحدة الحساب والمنطق ALU

٢- وحدة التزامن والتحكم

٣- مجموعة عدادات ومسجلات

ستتناول بالشرح المختصر وحدة الحساب والمنطق ALU، ووحدة التزامن والتحكم. أما المسجلات والعدادات فسوف



شكل ٨-٢ رسم توضيحي لوظيفة وحدة الحساب والمنطق

نشرحها بالتفصيل حيث أنها أهم المكونات التي يتعامل معها المستخدم عندما يقوم ببرمجة وحدة المعالجة المركزية أو المعالج.

#### ١- وحدة الحساب والمنطق ALU

من اسم هذه الوحدة نفهم الوظيفة المطلوبة منها وهي إجراء العمليات الحسابية والمنطقية. العمليات الحسابية مثل الجمع والطرح (وفي بعض المعالجات المتقدمة الضرب والقسمة وعمليات أخرى)،

والعمليات المنطقية مثل عمليات الأند، والأور، والنفي، وغيرها. في العادة تقوم وحدة التحكم بإرسال نوع العملية والمعاملات التي ستجرى عليها العملية إلى وحدة الحساب والمنطق التي بدورها تقوم بتنفيذ العملية وإرسال النتيجة إلى مسجل التراكم في معظم الأحيان. من الواضح أن هذه الوحدة تعامل مع إشارات بيانات، وبالتالي فإن عدد بتات هذه الوحدة سيساوى عدد مسار البيانات في المعالج، ففى المعالجات ذات ٨ بت ستكون وحدة الحساب والمنطق ٨ بت، وفي المعالجات ٦٤ بت ستكون هذه الوحدة ٦٤ بت أيضاً. هذا هو كل ما يكتفى أن نعرفه في هذا الموضوع حيث أننا نرى أنه ليس هناك ضرورة للشرح التفصيلي لتركيبها الداخلى حيث أنه يمكن من يريد المزيد عن هذه المعلومات أن يجد ذلك في معظم كتب الدوائر المنطقية أو الإلكترونيات الرقمية. شكل ٨-٢ يبين رسمًا توضيحيًا لوظيفة هذه الوحدة.

## ٢- وحدة التزامن والتحكم

وحدة التزامن هي الوحدة المسئولة عن إجراء أي فعل يقوم به المعالج بالتزامن أو التوافق مع نبضات الساعة الخاصة بهذا المعالج، تماماً مثلما يتواافق المتدربون في برامج تمارين الصباح مع المدرب وهو يقول كلمات إيقاعية مثل واحد، اثنين، واحد، اثنين، وهكذا حتى يتواافق الفريق معه في كل حركة يقوم بها، وكل كلمة ينطق بها لضبط الإيقاع. لذلك إذا كان إيقاع المدرب سريعاً فإن أداء الفريق سيكون سريعاً والعكس صحيح، ونفس المنطق يكون مطبقاً على أداء المعالج أو وحدة المعالجة المركزية CPU، فمع زيادة سرعة نبضات التزامن ترداد سرعة المعالج في أداء عملياته. وبالتالي فإن سرعة المعالج تتوقف في الأساس على سرعة نبضات الساعة. سنكتفى أيضاً بهذا القدر في الكلام عن وحدة التزامن والتحكم حيث لا يوجد تعامل مباشر بينها وبين المستخدم أو المبرمج.

## ٣- المسجلات والعدادات في شريحة المعالج

تستخدم المسجلات للتخزين المؤقت للمعلومات في صورة خانات ثنائية في داخل شريحة المعالج لحين الحاجة إليها. إن أي مسجل إزاحة يمكن تصميمه ليكون قادرًا على أداء الوظائف التالية:

١. إدخال المعلومات بالتوازي وإخراجها بالتوازي (سواء من الشمال لليمين أو من اليمين للشمال).
٢. دوران المعلومات في أي اتجاه وعكسه.
٣. إدخال المعلومات بالتوازي وإخراجها بالتوازي.
٤. إدخال المعلومات توالي من أي اتجاه وإخراجها توازي أو العكس.

المسجلات داخل المعالج يمكن النظر إليها على أنها واحد من نوعين، الأول هو مسجلات عامة الأغراض general purpose registers وهذه تستخدم في الكثير من الأغراض وتؤدي أكثر من وظيفة وعادة تكون هذه المسجلات متاحة للمستخدم لكي يتعامل معها، إما أن يسجل فيها أو يقرأ منها. النوع الثاني من المسجلات هو مسجلات خاصة للأغراض dedicated registers وهذه مسجلات موجودة لأداء غرض أو وظيفة واحدة لا تhind her عندها لخدمة أداء المعالج، وليس للمستخدم أي وسيلة للتحكم فيها سواء بالقراءة منها أو الكتابة فيها.

تستخدم العدادات counters عادة لعد النبضات الدخالة إليها ويمكن توظيف هذه العدادات لكي تقوم بعملية العد إما تصاعدياً أو تناظرياً مع ملاحظة أن خرج العدادات يكون دائمًا توازي. سنعرض فيما يأتي بشكل عام لوظيفة كل مسجل من المسجلات الرئيسية في وحدة المعالجة المركزية CPU وذلك دون تخصيص معالج معين لأن ذلك مطبق على جميع المعالجات تقريباً مع بعض الاختلافات البسيطة.

### ٣-١ مسجل التراكم, A Accumulator

يعتبر مسجل التراكم، وعادة يرمز له بالرمز A، من أكثر مسجلات المعالج عملاً ولذلك فإنه يمكننا النظر إليه على أنه سكريتيراً عاماً لشريحة المعالج. إن أي عملية حسابية أو منطقية يقوم بها المعالج لابد وأن يكون مسجل التراكم طرفاً فيها (وهذا بالذات في المعالجات ٨ بت)، فمثلاً لو أردت أن تجمع أي رقمين فإن واحداً منها لابد أن يوضع في مسجل التراكم وأما الرقم الآخر فيوضع في أي مسجل آخر أو حتى في الذاكرة. ليس هذا فقط بل إن نتيجة أي عملية حسابية أو منطقية لا توضع إلا في مسجل التراكم ومنه يمكن نقلها لأي مكان آخر وذلك في المعالجات ٨ بت. هناك مهمة أخرى أيضاً لهذا المسجل وهي أن أي عملية إدخال أو إخراج من خلال بوابات الإدخال أو الإخراج عادة تكون من خلال هذا المسجل. أي أن المعلومة توضع في مسجل التراكم أولاً ثم يتم إخراجها إلى بوابة الإخراج، أو إذا كانت المعلومة قادمة من بوابة إدخال فإنها توضع أولاً في مسجل التراكم ثم يتم نقلها منه لأي مكان آخر في داخل المعالج أو خارجه. إذن ما رأيك الآن في تسميتها بـ مسجل التراكم؟ إن عدد البتات (الخانات) bits الموجودة في مسجل التراكم دائماً يساوي عدد خطوط مسار البيانات bus data حيث أن كل تعاملاته تكون مع بيانات، ومن الممكن في بعض المعالجات أن يكون هناك أكثر من مسجل تراكم واحد. بعض هذه الوظائف الخاصة بالمركم يتم الاستغناء عنها في المعالجات ١٦ بت مثلاً حيث يمكن مثلاً في هذه المعالجات جمع رقمين من أي مسجلين لا يكوناً منهما المركم، كما أن النتيجة في هذه الحالة ليس من الضروري أن تذهب للمركم ولكنها تذهب لأحد هذين المسجلين.

### ٣-٢ عدد البرنامج Counter, PC

كما علمنا فإن مهمة المعالج الأساسية هي إحضار الأوامر من الذاكرة الواحد بعد الآخر ثم تنفيذها، ولذلك فإنه لابد لهذه المهمة من تحديد للأماكن التي تحتوي هذه الأوامر في الذاكرة. يحتوى عدد البرنامج دائمًا على عنوان المكان في الذاكرة الذي يحتوى الأمر الذي عليه الدور في التنفيذ، وكلما تم إحضار أي أمر من الذاكرة وقبل أن يتم تنفيذه فإن عدد البرنامج تتغير محتوياته بحيث تشير إلى عنوان الأمر القادم في التنفيذ. تذكر أيضاً أنه حتى لو حدث قفز من مكان في البرنامج إلى مكان آخر فإن وحدة التحكم داخل المعالج تضع عنوان الأمر الذي سيتم القفز إليه في عدد البرنامج حتى يصبح هو الأمر الذي عليه الدور في التنفيذ أيضاً حيث تنتقل عملية تنفيذ البرنامج إلى هناك. عدد برات هذا العداد دائمًا تساوي عدد برات مسار العنوان bus address وهذا منطقى جداً حتى يتمكن المعالج من إحضار الأوامر مهما كانت في أي مكان في الذاكرة سواء كانت في أولها أو في آخرها، لاحظ أن كمية الذاكرة التي يمكن أن يتعامل معها المعالج تتوقف على عدد البتات أو الخطوط في مسار العنوانين كما سنرى فيما بعد. إذا نظرنا إلى عدد البرنامج

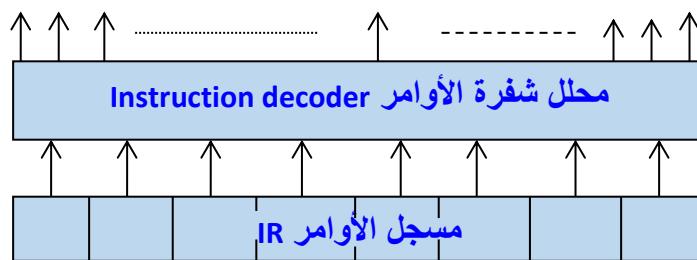
على أنه مسجل يحتوى عنوان الأمر الذى عليه الدور في التنفيذ فإننا سنصنفه على أنه من المسجلات ذات الأغراض الخاصة dedicated register الغير متاحة للمبرمج لاستخدامها في عمليات البرمجة.

### ٣-٣ مسجل ومحلل شفرة الأوامر Instruction Register, IR, And Decoder

بعد أن يتم إحضار الأمر من الذاكرة إلى شريحة المعالج لابد أن يسجل أو يوضع في أحد الأماكن في انتظار تنفيذه، هذا المكان هو مسجل الأوامر Instruction decoder, IR. أى أن مسجل الأوامر يحتوى شفرة الأمر الذى يتم تنفيذه الآن. لاحظ أن عدد برات مسجل الأوامر عادة يساوى عدد برات البايت في الذاكرة التي تساوى بدورها عدد برات مسار البيانات خاصة في هذا الجيل من المعالجات الذى نحن بصدده الآن (المعالجات ٨ بت)، كما أن عدد الأوامر التي يمكن للمعالج أن ينفذها سيتوقف على عدد البتات في مسجل الأوامر فمثلاً إذا كان عدد برات مسجل الأوامر هو ٨ بت فإن ذلك يعني أن هذا المعالج يستطيع التعامل مع  $2^8 = 256$  أمر على الأكثر.

أول خطوات تنفيذ أي أمر تبدأ من فاكلك أو محلل شفرة الأوامر instruction decoder الذي يتصل دخله بخرج مسجل الأوامر السابق كما في شكل ٩-٢ بحيث أنه على حسب شفرة الأمر الموجودة في مسجل الأوامر فإن عملية واحدة فقط سيتم تحليتها وتنفيذها على حسب الشفرة الموجودة على دخل محلل الشفرة ويتم ذلك بالطبع بمساعدة وحدة التحكم ووحدة الحساب والمنطق.

خطوط تنشيط للعمليات المختلفة داخل وحدة الحساب والمنطق ALU



شكل ٩-٢ رسم تخطيطي لمسجل ومحلل شفرة الأوامر

### ٤-٤ مسجل الحالة Status Register, SR

أحياناً يطلق على هذا المسجل اسم مسجل الأعلام Flag Register, FR. يعتبر هذا المسجل نشرة إخبارية تعكس حالة نتيجة آخر عملية حسابية أو منطقة قام المعالج بتنفيذها، فمن هذا المسجل نستطيع أن نعرف مثلاً إذا كانت هذه النتيجة سالبة أم موجبة أم تساوى صفرًا وغير ذلك من الأخبار المفيدة. هذا المسجل يحتوى على عدد من البتات وكل

واحدة منها تعتبر أو تسمى علما flag يعكس أو يدل على حالة معينة من العملية الحسابية أو المنطقية التي تم تنفيذها من هذه الأعلام ما يلى:

- **علم الصفر Zero flag, ZF** هذه البت تكون واحد إذا كانت نتيجة آخر عملية حسابية أو منطقية نفذها المعالج تساوى صفرًا وتكون هذه البت صفرًا إذا كانت النتيجة مختلفة عن الصفر سواء موجبة أو سالبة.
- **علم الإشارة Sign flag, SF** هذه البت تكون واحد إذا كانت نتيجة آخر عملية حسابية أو منطقية نفذها المعالج سالبة، أما إذا كانت هذه النتيجة موجبة فإن هذا العلم يكون صفرًا. لاحظ أن آخر بت في النتيجة تعكس إشارتها فإذا كانت آخر بت تساوى صفرًا فإن ذلك يعني أن النتيجة موجبة أما إذا كانت هذه البت واحدًا فإن ذلك يعني أن النتيجة سالبة لذلك فإنه دائمًا تكون محتويات علم الإشارة تساوى محتويات آخر بت في النتيجة.
- **علم الحمل Carry flag , CF** هذا العلم يكون واحد إذا حصل حمل carry من آخر بت في أي عملية جمع أو حصل استلاف Borrow لآخر بت في أي عملية طرح ويكون صفرًا إذا لم يكن هناك حمل أو استلاف في آخر عملية حسابية.
- **علم الباريقي Parity flag, PF** هذا العلم يكون واحد إذا كانت آخر عملية حسابية أو منطقية قام بها المعالج تحتوى على عدد زوجى من الوحайд أما إذا كانت هذه النتيجة تحتوى على عدد فردى من الوحайд فإن هذا العلم يكون صفرًا. ربما تسبب كلمة الباريقي بعض التشویش للقاريء، ولكن يكفى أن نعرف عنها في هذه الموضع أنها تستخدم في تصحيح أخطاء التراسل التتابع.
- **علم الحمل النصفى أو البيني HC** هذا العلم يكون واحدًا إذا كان هناك حمل من الخانة أو البت الثالثة إلى البت الرابعة نتيجة أي عملية جمع، أو هناك استلاف من البت الرابعة إلى البت الثالثة نتيجة أي عملية طرح، ويكون صفرًا فيما عدا ذلك أي إذا لم يحدث استلاف أو حمل من أو إلى البت الرابعة. لاحظ أننا هنا نبدأ عملية عد البتات بالرقم صفر، أي أن أول بت من ناحية اليمين هي البت رقم صفر. هذه الأعلام ستستخدم في أوامر القفز المشروط والنداء على البرامج الفرعية المشروطة كما يحدث عند البرمجة بلغة التجميع، لذلك يطلق على هذا المسجل أحياناً مسجل الشروط condition code register، كما أن كل من مسجل التراكم ومسجل الحالة يطلق عليهما **كلمة حالة المعالج Processor Status Word, PSW**.
- مسجل الأعلام قد يحتوى أعلاماً أخرى اعتماداً على نوع المعالج وهذه سيأتي تفصيلها عند الحاجة إليها في كل معالج أو كل متحكم على حده، ولكن دعنا الآن ننظر للمثال التالي كتطبيق سريع على هذه الأعلام.

**مثال ١**

اكتب محتويات الأعلام السابقة بعد إجراء عملية جمع الرقمين 77H و A5 . لاحظ أن الرقمين مكتوبين في الصورة السعشرية hexadecimal، وهذا هو المقصود من وضع الحرف H بعد كل رقم.

الجمع الثنائي للرقمين السابقين سيتم كما يلى:

0111 0111	الرقم الأول
1010 0101	الرقم الثاني
0001 1100	النتيجة

حمل 1

نلاحظ الآتى من النتيجة السابقة:

١. النتيجة لا تساوى الصفر، إذن فعلم الصفر يساوى صفر  $ZF=0$
٢. آخر بت في النتيجة صفر فالنتيجة موجبة وعلم الإشارة يساوى صفر  $SF$
٣. هناك حمل من البت السابعة (الأخيرة) فعلم الحمل يساوى واحد  $CF=1$
٤. النتيجة تحتوى ثلاثة وحايد (عدد فرد) فعلم الباريتى يساوى صفر  $PF=0$
٥. ليس هناك حمل من الخانة الثالثة للرابعة فعلم الحمل النصفى يساوى صفر  $HCF=0$

**٤-مسجل مؤشر المكذبة SP**

سيأتى شرحًا تفصيلياً للمكذبة stack فيما بعد في معرض الكلام عن البرامج الفرعية والمقاطعة، ولكن الآن بإمكانك أن تعرف أن المكذبة هي جزء من الذاكرة يتم فيه تخزين بعض العناوين أو البيانات المهمة والتي لابد من الحاجة إليها واسترجاعها مرة ثانية وبنفس الترتيب الذي تم تخزينها به. مسجل مؤشر المكذبة يحتوى عنوان آخر مكان تم التسجيل فيه في هذا الجزء من الذاكرة، لذلك فإنه طلما أن هذا المسجل سيحتوى على عنوان فلا بد أن يكون ١٦ بت. لاحظ أن المبرمج عادة وفي أغلب المعالجات تكون لديه الحرية في اختيار الجزء من الذاكرة الذى سيستعمل كمكذبة.

**٥-المسجلات عامة الأغراض General Purpose Registers**

في الكثير من الأحوال عندما نجمع أكثر من رقم، نحتاج لحفظ نتيجة معينة لحين استخدامها في عملية أخرى لاحقة، ولذلك فإنه بدلاً من إرسال هذه النتيجة إلى الذاكرة ثم استدعائها ثانية مما يأخذ الكثير من الوقت فقد تم تجهيز المعالج ببعض المسجلات التي تستخدم لتخزين مثل هذه النتائج المرحلية لحين الحاجة إليها. عدد البتات في هذه المسجلات

يكون عادة مساوياً لعدد برات مسار البيانات. عدد هذه المسجلات مختلف من معالج آخر ومن شركة لأخرى. ولقد تم التعارف على تسمية هذه المسجلات بالسجلات B و C و D و E و H و L كما سمى المركم من قبل بالسجل

مسجل التراكم A	مسجل الحالة F
B	C
D	E
H	L
مسجل الإشارة IX	
مسجل الإشارة IY	
مسجل المكدة SP	
عداد البرنامج PC	
مسجل إنعاش الذاكرة R	مسجل الأوامر IR

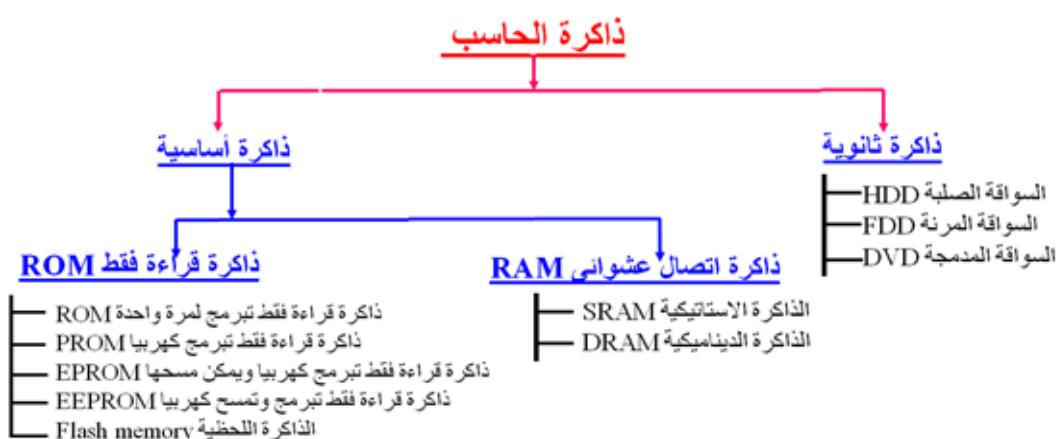
شكل ١٠-٢ الحد الأدنى للسجلات في معظم المعالجات ٨ بت يحتوى الواحد فيها على ٨ برات فقط. في هذه الحالة يكون كل سجل له سجل آخر يمكن ازدواجه معه ولا يمكن ازدواجه مع أي سجل آخر، فمثلاً السجل B لا يزدوج إلا مع السجل C فقط وكذلك السجل D لا يزدوج إلا مع السجل E والمسجل H لا يزدوج إلا مع السجل L. لاحظ أنه في حالة ازدواج المسجل B والمسجل C فإن المسجل C يحتوى أو يمثل البایت ذات القيمة الصغرى low significant byte من المعلومة المكونة من ١٦ بت والمسجل B يحتوى البایت ذات القيمة العظمى high significant byte من هذه المعلومة. بنفس الطريقة في حالة الأزواج DE و HL فإن المسجلات E و L تحتوى البایت ذات القيمة الصغرى والمسجلات D و H تحتوى البایت ذات القيمة العظمى. فمثلاً إذا أردنا أن نسجل المعلومة 4CF6H المكونة من ١٦ بت في زوج المسجلات HL فإن البایت F6 وهى البایت ذات القيمة الصغرى لابد أن توضع في المسجل L وأما البایت 4C ذات القيمة العظمى فتوضع في المسجل H. في شكل ١٠-٢ ستلاحظ أن هذه المسجلات موضوعة بنفس طريقة وكيفية ازدواجها. سجل إنعاش الذاكرة الموجود في شكل ١٠-٢ يوجد فقط في حالة المعالجات التي لديها القدرة على التعامل مع الذاكرة الديناميكية dynamic RAM مثل المعالج Z80، حيث أن هذه الذاكرة تحتاج إلى إعادة تجديد محتواها كل فترة زمنية معينة (أقل من ٣ ميلليثانوية) لأنها تكون عبارة مكتفات تستخدم كوحدات تخزين للواحد والصفر وليس ماسكات static RAM latches كما هو الحال في الذاكرة الاستاتيكية.

A. هذه التسمية كما سنرى هي التسمية التي ستستخدم مع لغة الأسبلى (التجميع) assembly language . شكل ١٠-٢ يبين جميع المسجلات التي تكلمنا عنها حتى الآن والتي تمثل كما ذكرنا الحد الأدنى لمحطويات أي معالج من المسجلات. هناك بعض الأوامر التي تعامل مع هذه المسجلات كأزواج يتكون كل زوج منها من ١٦ بت بدلاً من التعامل معها كمسجلات

## ٧-٢ أنواع الذاكرة

ستتكلّم كثيراً ونتعامل مع أنواع مختلفة من الذاكرة ولابد أن نكون على دراية بمواصفات وخصائص كل نوع من هذه الأنواع من الذاكرة التي يتعامل معها المعالج أو تكون موجودة داخل أي متحكم. بالطبع لن ندخل في التفاصيل الإلكترونية لكل نوع لأن ذلك من شأن كتب متخصصة في الإلكترونيات الرقمية، ولكننا كما قلنا سنتكلّم فقط عن مواصفات هذه الأنواع. شكل ١١-٢ يبيّن رسمياً تخطيطياً للأنواع المختلفة التي من الممكن أن يتعامل معها المعالج أو تكون موجودة داخل أي متحكم. في البداية يتم تقسيم الذاكرة إلى نوعين أساسيين وهما:

### الذاكرة الأساسية Main memory



شكل ١١-٢ تقسيمات ذاكرة الحاسوب

الذاكرة الأساسية هي الذاكرة التي يوضع بها أي برنامج تقوم وحدة المعالجة المركزية CPU بتنفيذـه. وقد أشرنا من قبل في معرض الحديث عن مسار العنوانـين في المعالج أن هذه الذاكرة يتـحدـد مقدارـها بمقدارـ عدد خطوطـ أو بـتات مـسارـ العنـوانـينـ. ولـقد رأـيـناـ أنـ مـقـدـارـ هـذـهـ الـذـاـكـرـةـ يـسـاوـيـ ٢ـ أـسـ عـدـدـ خـطـوـطـ مـسـارـ عـنـوانـينـ كـحـدـ أـقـصـىـ،ـ ولـذـلـكـ فـهـيـ تـسـاوـيـ ٦٤ـ جـيـجاـبـاـيـتـ فـيـ حـالـةـ الـمعـالـجـ بـنـتـيـوـمـ ٤ـ الـذـىـ لـهـ ٣٦ـ خـطـ عـنـوانـينـ.ـ ولـقدـ أـشـرـنـاـ مـنـ قـبـلـ أـيـضـاـ أـنـ الـحـاسـبـاتـ الشـخـصـيـةـ الـتـىـ لـدـيـنـاـ لـاـ تـحـتـوـيـ كـلـ هـذـاـ الـمـقـدـارـ مـنـ الـذـاـكـرـةـ الـأـسـاسـيـةـ فـهـيـ تـحـتـوـيـ حـوـالـيـ ٤ـ جـيـجاـبـاـيـتـ تـقـرـيـباـ وـذـلـكـ لـعـدـةـ أـسـبـابـ مـنـهـاـ أـنـ مـعـظـمـ الـبـرـمـيـاتـ لـاـ تـحـتـاجـ لـكـلـ هـذـاـ الـكـمـ مـنـ الـذـاـكـرـةـ بـالـإـضـافـةـ إـلـيـ أـنـ مـصـنـعـيـ الـحـاسـبـاتـ الشـخـصـيـةـ يـحـاـولـونـ

جعل أسعارها في صورة مثالية لإمكانيات أغلب المستخدمين. أما المستخدمين المتخصصين فقد يحتاجون إلى هذه الكمية القصوى من الذاكرة.

هذه الذاكرة الأساسية تقوم الشركات المصنعة للحواسيب بتقسيمها إلى جزأين وهما:

### ذاكرة الاتصال العشوائي RAM

هذه الذاكرة هي التي تحتوى البرامج والبيانات التي يتعامل معها المستخدم ويتم فقد محتواها بانقطاع القدرة عن نظام الحاسب (لذلك تسمى بالذاكرة المتطايرة). يتم بناء هذه الذاكرة إلكترونياً بطريقتين:

- الذاكرة الاستاتيكية static ram: هذه الذاكرة تكون كل بت من بتاتها عبارة عن ماسك latch، ولذلك فإنها تكون مرتفعة الثمن وزمن الاتصال بهذا النوع من الذاكرة يكون أقل.
- الذاكرة الديناميكية dynamic ram: هذه الذاكرة تكون وحدة بناؤها (كل بت من بتاتها) عبارة عن مكثف، بحيث عندما يكون المكثف مشحوناً فإنه يمثل إحدى الحالات المنطقية (الواحد مثلاً) وعندما يكون فارغ فإنه يمثل الحالات المنطقية الأخرى (الصفر مثلاً). من عيوب هذه الطريقة أن المكثف يفقد شحنته في خلال زمن يساوى تقريباً ٣ إلى خمسة ميلليثانبيه. لذلك فإن هذه الذاكرة يصاحبها دائماً دائرة كهربائية تحدد محتواها باستمرار (تعيد الكتابة في المكثف) قبل أن يفقد المكثف هذه المحتويات. بالطبع فإن هذه الذاكرة تكون أرخص كثيراً عن الذاكرة السابقة، ولذلك فإنها تستخدم عادة في بناء ذاكرة الحاسوب العشوائية RAM. شكل ١٢-٢ يبين صورة لوحدات من هذه الذاكرة التي يتم تركيبها على اللوحة الأم للحاسوب.



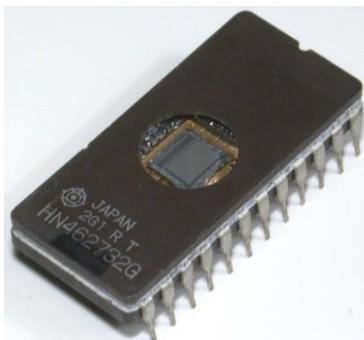
شكل ١٢-٢ وحدات الذاكرة RAM التي يتم تركيبها على الحاسوب

### ذاكرة القراءة فقط ROM

هذه الذاكرة تميز بأنها لا تفقد محتواها بانقطاع القدرة عنها (غير متطايرة)، ولذلك فإن الذاكرة الأساسية لأى حاسب لابد أن يكون جزءاً منها من هذا النوع حيث يتم عليها تسجيل ثوابت الحاسوب وجدائل البحث وغير ذلك من البيانات التي لابد من تواجدها عند بداية تشغيل الحاسب. أيضاً فإن كل النظم الكامنة أو المدمجة تعتمد على هذا النوع

من الذاكرة حيث يتم عليها تسجيل برنامج التحكم في النظام الذي من أهم صفاتة أنه يكون ذاتي التشغيل، أى أنه يعمل بمجرد توصيل القدرة للنظام، وهذا ما سنراه في هذا المقرر. هذه الذاكرة توجد في عدة أنواع منها ما يلى:

- **الذاكرة ROM التي تبرمج مرة واحدة:** وهي ذاكرة يوضع عليها برنامج النظام المدمج مثل برنامج تشغيل الغسالة أو المصعد مثلاً، وهي تبرمج مرة واحدة فقط عن طريق مصنع النظام بحيث لا يمكن إعادة برمجتها أو تعديل محتواها على الإطلاق ولذلك فإنها تستخدم في حالة الإنتاج بالجملة. تتميز هذه الذاكرة بشخص ثانٍ وهي تسمى عادة ذاكرة البرمجة مرة واحدة OTP, One Time Program.
- **الذاكرة ROM التي تبرمج كهربيا:** Programable ROM, PROM: وهي نفس الذاكرة السابقة تقريباً ولكنها يمكن برمجتها كهربياً بتمرير تياراً كهربياً يحرق فيوزات (منصهرات) بداخلها، ويتم ذلك بوضعها في جهاز خاص بذلك وهي تتميز أن المستخدم يمكنه برمجة هذه الشرائح باستخدام هذا الجهاز، ولكنها لازالت تبرمج مرة واحدة فقط لأنها لا يمكن مسح محتواها أو تعديل محتواها.
- **الذاكرة ROM التي تبرمج كهربياً ويمكن مسحها:** Erasable Programable ROM, EPROM: هذه الذاكرة يمكن برمجتها بوضعها في جهاز خاص بالبرمجة، كما يمكن مسح محتواها بوضعها في جهاز آخر خاص بالمسح حيث يتم تعریضها لأشعة فوق البنفسجية من خلال نافذة زجاجية لمدة ١٥ إلى ٢٠ دقيقة وبذلك يمكن إعادة برمجتها كما في شكل ١٣-٢. لاحظ أنه لا يمكن التعامل مع أجزاء معينة من هذه الشرائح، أى أنها عندما تمسح فإن كل محتواها يتم مسحها ولا يمكن التعامل مع جزء معين منها.



شكل ١٣-٢ شريحة ROM يتم مسح محتواها بالأشعة فوق البنفسجية

بشكل كبير، وهي في الحقيقة نوع خاص من الذاكرة السابقة EEPROM التي تبرمج وتحسّن كهربياً ولا تحتاج لأنشدة فوق البنفسجية لمسح محتواها ولكن محتواها تمسح أيضاً بتمرير تيار كهربائي خلالها. لذلك تتميز هذه الذاكرة

- **الذاكرة ROM التي تبرمج وتحسّن كهربياً:** Electrically Erasable Programable ROM, EEPROM: وهذا النوع يمكن مسح محتواه كهربياً دون الحاجة لتعریضها لأشعة فوق البنفسجية، ولذلك فإنها يمكن برمجتها وهي في نفس مكانها في التطبيق الملحق به.
- **الذاكرة ROM اللحظية:** وهي ما يسمى flash memory التي ظهرت في السينين الأخيرة وانتشرت

بإمكانية برمجتها ومسحها وهى في نفس مكانها في النظام، بل يمكن تعديل محتواها أيضاً وهي في نفس مكانها كما نرى مع الذاكرة الحيوية التي تتعامل معها بكثرة هذه الأيام.

### الذاكرة الثانوية Secondary memory

النوع الثاني من ذاكرة الحاسوب هو الذاكرة الثانوية وهذه لا دخل لها بمسار عناوين ووحدة المعالجة المركزية CPU وهذه تكون موجودة في صور عديدة منها ما هو موضح في شكل ١١-٢ حيث من أمثلة ذلك السوقة الصلبة Hard Disk Drive، السوقة المرنة Floppy Disk Drive، FDD، Disk Drive، HDD، والسوقة المدمجة Digital Video Disk، DVD التي يتم التسجيل عليها والقراءة منها بطريقة مغناطيسية. هناك أيضاً السواقات المدمجة التي يتم التسجيل عليها والقراءة منها بطرق ضوئية. بالطبع فإن كل هذه الطرق لا مكان لشرح تفصيلي لها أكثر من ذلك لأنها ليس هذا هو المكان المناسب لذلك.

### ملخص الفصل

وحدة المعالجة المركزية CPU هي أحد المكونات الأساسية في جميع المتحكمات، ولذلك لزم أن نلقي نظرة تفصيلية على أطرافها وهيكلها الداخلي. حيث تعرفنا من خلال ذلك على معنى مسار العناوين، ومسار البيانات، وخطوط التحكم والسمات المميزة لكل منها. ثم فتحنا شريحة وحدة المعالجة المركزية وتعرفنا على هيكلها الداخلي بما يحتويه من وحدة الحساب والمنطق، ووحدة التحكم والتزامن، ومجموعة المسجلات والعدادات الشائعة في أي واحدة من هذه الوحدات. ثم تطرق الحديث على أنواع الذاكرة وتصنيفاتها المختلفة.

## الفصل ٣

### نظرة شاملة على المتحكمات AVR

### Overview on AVR Microcontrollers

**العناوين المضيئة في هذا الفصل:**

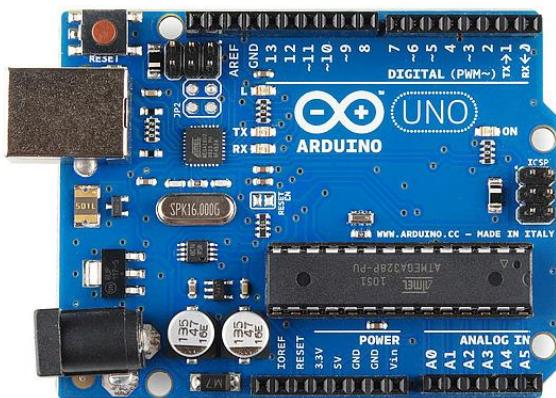
- ١- مكونات المتحكمات AVR
- ٢- تنفيذ الأوامر في المتحكمات AVR
- ٣- البناء المعماري CISC و RISC
- ٤- تداول البيانات داخل المتحكمات AVR
- ٥- نظرة موسعة على منتجات أتمل من المتحكمات AVR

## ١-٣ مقدمة

**سنلقي** في هذا الفصل نظرة شاملة على المتحكمات AVR المصنعة من قبل شركة Atmel والتي تم اختيارها لتكون هي عائلة المتحكمات المصاحبة لهذا الكتاب، وسيتم التركيز على المتحكم atmega328 بالذات حيث أنه هو المتحكم المصاحب لنظام الأردوينو الشهير الذي شاع استخدامه هذه الأيام. لاشك أن استخدام نظام الأردوينو لأحد شرائح المتحكمات AVR كان من العوامل المهمة جداً في شهرة هذه العائلة من المتحكمات والإقبال عليها من قبل الكثير من المستخدمين. إن ذلك يذكرنا تماماً بما حدث من شركة IBM التي أقدمت في بداية الثمانينيات من هذا القرن على استخدام المعالج intel8086 في بناء نظام الحاسوب الشخصي PC الذي كان باكورة الحاسوب الشخصية في هذا الوقت، والذي كان بمثابة الشارة لثورة الحاسوب التي نعيشها الآن وبالتالي الشهرة الواسعة لم المنتجات شركة intel في مجال المعالجات التي نراها الآن. بالإضافة لذلك فإن هذه العائلة من المتحكمات تتمتع بالسرعة، وكثرة الملحقات التي تسهل من استخدامها في الكثير من التطبيقات، والاستهلاك الأقل للقدرة، وإتاحة الكثير من شرائحها بامكانيات وأحجام مختلفة، ولا ننسى إمكانية البرمجة بلغة التجميع assembly language

الخاصة بها أو بلغة C العالية المستوى، وأيضاً إتاحة شركة أتمل المصنعة للكثير من المواد المساعدة والتي من أهمها وسط البرمجة المتكمال أقل استديو Atmel studio كما سنرى بالتفصيل في هذا الكتاب.

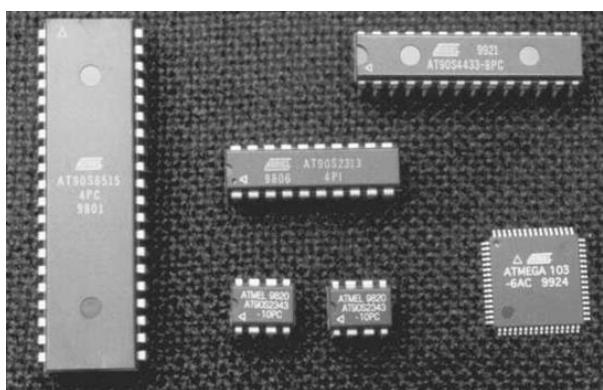
ربما يقول البعض، إنني من محترف الأردوينو، وأستطيع أن أعمل به ماشت من مشاريع، فلماذا نضيع وقتنا في تعلم المتحكمات؟ لكنني أجيب على هذا السؤال لابد أن نعطي فكرة عن ما هو الأردوينو وما هي مميزاته وعيوبه؟ الأردوينو باختصار هو نظام كامل على لوحة إلكترونية واحدة كما في شكل ١-٣



شكل ١-٣ شكل عام للوحدة الأردوينو

تحتوي المتحكم atmega328 (الذى سندرسه بالتفصيل في هذا الكتاب)، بالإضافة إلى بعض الإلكترونيات الإضافية بحيث يجعل هذا النظام سهل التعامل مع الكثير من التطبيقات الخارجية التي يمكن قراءة بيانات منها مثل المحسسات المختلفة، أو إرسال بيانات لتشغيلها مثل المواتير ولبات الإضاءة، بطريقة مناسبة للهواة تعينهم عن الخوض في تفاصيل المتحكمات. ولقد كان وراء هذه الفكرة مجموعة من الطلاب الإيطاليين الذين فكروا في هذه الفكرة لتكون بمثابة أداة مساعدة في تصميم الروبوتات للهواة. وكان هؤلاء الطلاب يتقابلون لشرح فكرتهم في بار بأحد المدن الإيطالية وهذا البار كان اسمه أردوين Arduin وهو أحد الحكم العسكريين في إيطاليا من عام ٢٠٠٢ حتى عام

١٠١٤ (ومن هنا كانت التسمية أردوينو Arduino)، ولقد بدأ هؤلاء الطالب هذا المشروع في عام ٢٠٠٣ تقريبا.



شكل ٢-٣ إصدارات مختلفة من المتحكمات AVR

سعر الوحدة من هذه اللوحة الآن (مايو ٢٠١٧) هو ١٧ دولار تقريبا. سعر شريحة المتحكم هو atmega328 قلب نظام الأردوينو حوالي ١٥ دولار إلى ٣ دولار تقريبا على حسب سرعة التزامن. بفرض أنك ستقوم ببناء نظام كامن أو مدمج embedded system، وهذا النظام ستقوم بتصنيع أعداد كبيرة منه، فهل من الأفضل أن تستخدم لوحة أردوينو أم تستخدم شريحة متحكم كالمبيبة في شكل ٢-٣، مع العلم أن طريقة البرمجة هي نفسها تقريبا وهي لغة C

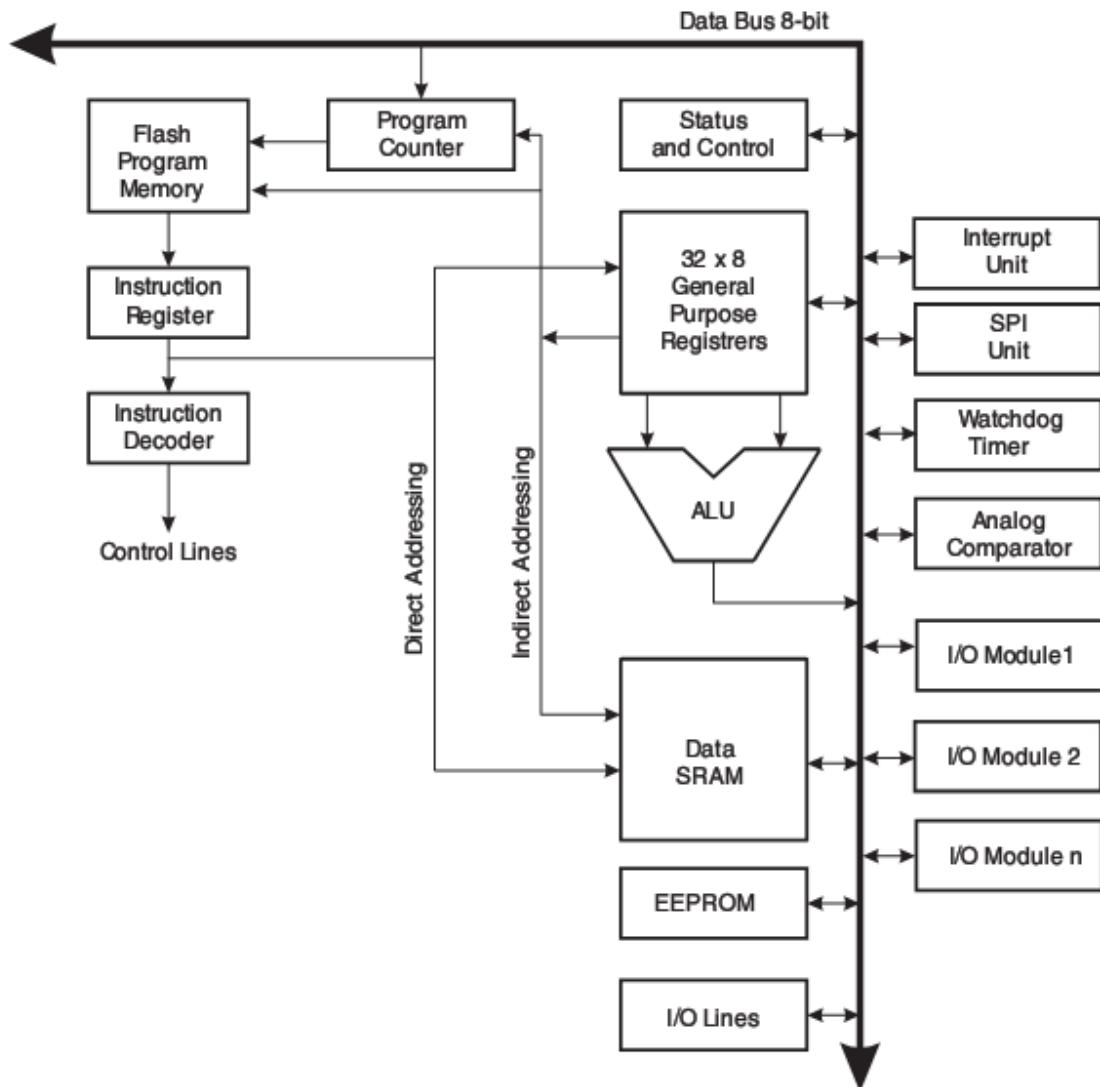
أو أحد صورها. إذن سيكون هناك توافراً مادياً كبيراً عند استخدام المتحكم مباشرة في أي مشروع عن استخدام لوحة الأردوينو التي تكون مزودة بكثير من الإمكانيات البرمجية software، والمكونات hardware، التي لا تكون ذات فائدة وزائدة عن الحاجة عند الاستخدام في تصميم النظم الكامنة. هذا بالإضافة إلى أن المتحكمات تكون موجودة في الكثير من الإصدارات ذات الإمكانيات المختلفة بحيث يمكنك اختيار المتحكم الأكثر ملائمة لتطبيقك والأقل سعراً، فهناك مثلاً المتحكمات ATiny التي لها ثمانية أرجل فقط وذات إمكانيات وسرعات معقولة تناسب الكثير من التطبيقات وبالتالي فهي أرخص سعراً. إذن نظام الأردوينو مناسب جداً لأغراض تعليم الهواة الذين ليس لديهم فكرة إلكترونية كافية والذين يصممون مشاريعاً أو نظماً أحادية فقط. أما من يريد تعلم تصميم النظم الكامنة embedded systems بغرض الإنتاج على المستوى التجاري فليس أمامه مناص من تعلم المتحكمات وأساليبها، وهذا لن يكون صعباً على الإطلاق لمن لديه فكرة بسيطة عن أساسيات الدوائر المنطقية وأساسيات البرمجة.

## ٢-٣ مكونات المتحكمات AVR

شكل ٣-٣ يبين رسمياً صنديقياً للمكونات الداخلية (architecture) للمتحكمات AVR بصورة عامة، ويستكمل باختصار عن كل واحد من هذه المكونات. بالنظر لهذه المكونات سنجد أنها إما أن تكون مكونات تخص وحدة المعالجة المركزية CPU، التي سبق شرحها في الفصل السابق، أو أنها مكونات ملحقة داخل شريحة المتحكم كما ذكرنا في الفصل ٢.

## وحدة الحساب والمقطق ALU

وهذه من المكونات الأساسية لوحدة المعالجة المركزية CPU ووظيفتها هي إجراء العمليات الحسابية والمنطقية.



شكل ٣-٣ رسم صندوقى للمكونات الداخلية للمتحكمات AVR (صورة من دليل المتحكمات AVR)

## عداد البرنامج Program Counter, PC

وهو عداد تشير محتوياته إلى عنوان الأمر الذي عليه الدور في التنفيذ كما رأينا في الفصل ٢ . عدد بتاته سيتوقف على كمية ذاكرة البرمجة الموجودة في شريحة المتحكم، وكما نعلم فإن كمية هذه الذاكرة تختلف على حسب إصدار المتحكم، فمثلاً المتحكم atmega328 به ٣٢ كيلوبايت من الذاكرة اللحظية غير المتطابقة (تحتفظ بمحطياتها حتى

بعد انقطاع القدرة) والقابلة للمسح والبرمجة حتى وهي في مكانها في التطبيق أو على اللوحة الإلكترونية In system self-programmable flash memory. معنى ذلك أن عدد برات عداد البرنامج في المتحكم atmega328 سيكون ١٥ بت (لأن ٢ أُس ١٥ يساوى ٣٢ كيلو) وبالتالي يمكن للعداد أن يتعامل أو يشير على أي أمر في أي مكان في هذه الذاكرة.

### مسجل الأوامر Instruction Register

هذا المسجل أيضاً من مكونات وحدة المعالجة المركزية CPU وهو يحتوى شفرة الأمر الذى يتم تنفيذه الآن. وعدد براتته يساوى عدد برات مسار البيانات (٨ بت في حالة المتحكمات ٨ بت). لاحظ اتصال هذا المسجل بذاكرة البرامح اللحظية كما في شكل ٣-٣.

### محلل شفرة الأوامر Instruction decoder

هذا المخلل يتصل بخرج مسجل الأوامر حيث سيقوم هذا المخلل بفك شفرة الأوامر وبناء على ذلك تنشيط خطوط التحكم المناسبة لكل أمر.

### مسجل الحالة Status Register, SR

وهو أحد مسجلات وحدة المعالجة المركزية داخل المتحكم، ويحتوى مجموعة الأعلام التي تبين حالة آخر عملية حسابية أو منطقية تم تنفيذها عن طريق وحدة الحساب والمنطق. شكل ٤-٣ يبين رسمياً تخطيطياً للأعلام الموجودة في هذا المسجل. الشكل ٤-٤ يبين عنوان هذا المسجل وهو 0x3F، ( هنا تعنى أن الرقم مكتوب بنظام العد الثنائى) ويبيّن أيضاً أن جميع محتويات هذا المسجل تكون أصفاراً عند إعادة تشغيل المتحكم أو توصيل القدرة له. من هذه الأعلام أربعة أعلام نعرفهم من الفصل السابق في معرض الحديث عن مسجل الأعلام في وحدة المعالجة المركزية CPU وهم كالتالى:

#### ١- علم الحمل أو الاستلاف C

هذا العلم هو البت رقم صفر في مسجل الحالة، وهو يساوى ١ عند حدوث حمل من آخر بت (رقم ٧ حيث أننا نبدأ عد البتات بالبت رقم صفر) في حالة الجمع أو استلاف إلى آخر بت في حالة الطرح. بالطبع مع العمليات المنطقية مثل الآند والأور وغيرها لن يكون هناك حمل أو استلاف وبالتالي سيكون هذا العلم يساوى صبراً بعد جميع العمليات المنطقية. هذا العلم هو البت رقم صفر في مسجل الحالة.

العنوان 0x3f

7	6	5	4	3	2	1	0
I	T	H	S	V	N	Z	C
0	0	0	0	0	0	0	0

### القيم التلقائية عند إعادة الوضع

شكل ٣-٤ مسجل الحالة في المتحكمات AVR

#### ٢-علم الصفر, Z

هذا العلم هو البت رقم ١ في مسجل الحالة، وهو يساوى واحد إذا كانت نتيجة آخر عملية حسابية أو منطقية تم تنفيذها عن طريق وحدة الحساب والمنطق ALU تساوى صفر. ويساوى صفر فيما عدا ذلك.

#### ٣-علم السالبة N

هذا العلم هو البت رقم ٢ في مسجل الحالة، وهو يساوى آخر بت في النتيجة، فإذا كانت آخر بت تساوى صفر، فإن ذلك يعني أن النتيجة موجبة، وإذا كانت آخر بت تساوى واحد، فإن ذلك يعني أن النتيجة سالبة، لذلك سمى بعلم السالبة لأنه عندما يكون واحد فإن ذلك يعني أن النتيجة سالبة. هناك ارتباطاً وثيقاً بين هذا العلم وعلم الفيضان V وعلم الإشارة S فيرجى النظر إلى الجزء التالي الخاص بالفيضان.

#### ٤-علم الفيضان V

هذا العلم هو البت رقم ٣ في مسجل الحالة، وهو يبين الفيضان على علم السالبة N في حالة الجمع أو الطرح كما سنرى في الجزء التالي الخاص بالفيضان.

#### ٥-علم الإشارة S

هذا العلم هو البت رقم ٤ في مسجل الحالة، وهو عبارة عن عملية الإكس أور XOR لمحتويات علم الفيضان V وعلم السالبة N. وهو يبين الإشارة الصحيحة للنتيجة إذا حدث فيضان على علم السالبة وتسبيب في تدميره، وسنبين ذلك في الجزء التالي الخاص بالفيضان أيضاً.

**٦-علم الحمل النصفي H**

هذا العلم هو البت رقم ٥ في مسجل الحالة، وهو يساوى واحد إذا كان هناك حملا من البت الثالثة إلى البت الرابعة في حالة الجمع، أو استلاف من البت الرابعة إلى البت الثالثة في حالة الطرح، ويكون صفرًا فيما عدا ذلك، وبالطبع فإنه سيكون صفرًا في حالة العمليات المنطقية.

**٧-علم نسخ أو تخزين بت T Bit copy storage, T**

هذا العلم هو البت رقم ٦ في مسجل الحالة. إفترض أننا نريد نسخ البت الثالثة في المسجل R1 (أحد المسجلات العامة) في البت رقم ٥ مثلاً في المسجل R2 (أحد المسجلات العامة أيضًا). لا يوجد هناك أمر من أوامر لغة تجميع المتحكمات AVR يقوم بهذه العملية مباشرة، ولذلك وجد العلم T. في هذه الحالة يتم استخدام الأمر BLD R1,3 الذي يقوم بنسخ (Bit Load, BLD) البت ٣ في المسجل R1 في العلم T، ثم بعد ذلك يتم استخدام الأمر BST R2,5 الذي يقوم بنسخ (Bit Store, BST) العلم T في البت رقم ٥ في المسجل R2، وبذلك يتم نقل محتويات أى بت في أى مسجل إلى أى بت في أى مسجل آخر عبر هذا العلم T. كل من هذين الأمرين هما من أوامر لغة تجميع المتحكمات AVR.

**٨-علم تنشيط المقاطعة العام I Global interrupt enable flag, I**

هذا العلم هو البت رقم ٧ في مسجل الحالة. وضع هذا العلم يساوى واحد من قبل المستخدم ينشط جميع مصادر المقاطعة في المتحكم، ووضعه يساوى صفر يخمد أو يمنع جميع مصادر المقاطعة. أى واحد من المتحكمات AVR يكون له العديد من مصادر المقاطعة كما سنرى في الفصول المختلفة من هذا الكتاب، وكل مصدر من هذه المصادر يكون له علم تنشيط آخر خاص به، ولكنكي يتم تنشيط أى مصدر من هذه المصادر لابد أن يكون علم المقاطعة العام I يساوى واحد أولاً، ثم نجعل علم التنشيط الخاص بهذا المصدر يساوى واحد أيضاً، وسيوضح ذلك بالتفصيل مع الحديث عن مصادر المقاطعة المختلفة.

**Overflow الفيضان**

الأرقام التي نتعامل معها في العمليات الحسابية إما أن تكون أرقاماً كليلة، أى ليس لها إشارة وبالتالي فإن كل بذات الرقم تعبر عن هذا الرقم. فلو فرضنا أننا نتعامل مع أرقام من بait واحدة (٨ بت)، فإن ذلك يعني أن أكبر رقم ثنائى في هذه الحالة سيكون ١١١١١١١١ وهو ما يساوى ٢٥٥ في النظام العشري، وأصغر رقم سيكون ٠٠٠٠٠٠٠٠ وهو ما يساوى الصفر في النظام العشري. لكنكي يتم التعبير عن الأرقام السالبة أو ذات الإشارة على وجه العموم، فإن هناك أكثر من نظام لذلك وأشهرها هو نظام المتمم الثنائي للرقم (راجع كتاب الدوائر المنطقية للمؤلف). لو

فرضنا مثلاً الرقم 00000101=5، سيكون متممه الثنائي هو الرقم 5- كما يلى: 11111011=5-. نلاحظ من ذلك أن الأرقام الموجبة ستكون آخر بت فيها (البت رقم 7) تساوى صفر، بينما الأرقام السالبة ستكون آخر بت

أرقام بدون إشارة		أرقام ذات إشارة	
ثانية عشرى	ثانية عشرى	ثانية عشرى	ثانية عشرى
255	11111111	+127	01111111
-----	-----	-----	-----
128	10000000	+1	00000001
127	01111111	0	00000000
-----	-----	-1	11111111
2	00000010	-2	11111110
1	00000001	-----	-----
0	00000000	-128	10000000

شكل ٣-٥ تمثيل الأرقام السالبة والموجبة بنظام

المتمم الثنائى

فيها تساوى واحد. على سبيل المثال الرقم 11111100 يمثل رقم سالب لأن آخر بت تساوى واحد، ولذلك نعرف قيمة هذا الرقم يوجد المتمم الثنائى لهذا الرقم وهو: 00000100، وبالتالي فإن هذا الرقم يساوى 4-. أصغر رقم موجب سيكون 00000000 وأكبر رقم موجب سيكون 01111111 وهو ما يساوى 127. أما أكبر رقم سالب فسيكون كالتالى: 11111111-1، بينما أصغر رقم سالب في هذه الحالة سيكون 10000000-128. شكل ٣-٣ يبيّن تمثيل الأرقام ذات الإشارة والتي ليس لها إشارة.

عمليات الجمع والطرح على الأرقام التي ليس لها إشارة ليس بها

بها أى مشكلة لأننا نقوم بعملية الجمع المنطقى حتى لو حصل حمل من الخانة الأخيرة، فإن هذا الحمل يعتبر من ضمن الرقم لأن الإشارة هنا ليس لها اعتبار. والمثال التالى يوضح ذلك:

$$\begin{array}{r} 127 \\ + 1 \\ \hline 128 \end{array} \quad \begin{array}{r} 01111111 \\ + 00000001 \\ \hline 10000000 \end{array}$$

في هذا المثال ليس هناك أى تمثيل للإشارة لذلك فإن مجموع الرقم الأول وهو 127 زائد الرقم الثانى وهو 1 كانت النتيجة 128 وتم اعتبار النتيجة تساوى ١٢٨ لأنه كما ذكرنا بأنه لا اعتبار للإشارة هنا.

في حالة اعتبار الإشارة في نفس المثال السابق، فإن الرقم الأول هو 127+ والرقم الثانى 1+ وبجمعهما كانت النتيجة 10000000، والمفاجأة هنا أن آخر بت في النتيجة تساوى 1 مما يعني أن هذه النتيجة سالبة، أي أنه تم جمع رقم موجب (127+) مع رقم موجب آخر (1+) فكانت النتيجة سالبة، وهذا هو ما يسمى بخطأ الفيضان. المشكلة هنا أنك لو تبعـت عملية الجمع بت بعدـت ستـجد أنه قد حصل حـل على آخر بت، وهذا هو السبـب في الخطـأ الذي حصل في بت الإشارة. في هذه الحـالة ستـقوم وحدـة المعـالـجـة المـركـزـية CPU بوضع علمـ الفـيـضـان V=1 دلـلة علىـ أنـ هـنـاكـ خـطـأـ فيـ الإـشـارـةـ،ـ وـأـنـ هـذـهـ النـتـيـجـةـ لـيـسـ النـتـيـجـةـ الصـحـيـحةـ.ـ تـذـكـرـ أـنـ عـلـمـ السـالـيـةـ Nـ الذـىـ يـسـاـوىـ آـخـرـ بتـ فـيـ النـتـيـجـةـ سـيـكـوـنـ 1=Nـ.ـ عـلـمـ الإـشـارـةـ Sـ كـمـ ذـكـرـنـاـ فـيـ تـعـرـيفـهـ بـأنـهـ يـسـاـوىـ عـلـمـ الإـلـكـسـ أـورـ عـلـىـ كـلـ منـ عـلـمـ السـالـيـةـ Nـ وـعـلـمـ الفـيـضـانـ Vـ،ـ وـحـيـثـ أـنـ كـلـ مـنـهـمـ يـسـاـوىـ 1ـ فـإـنـ نـتـيـجـةـ الإـلـكـسـ أـورـ سـتـعـطـىـ S=0ـ،ـ وـهـذـهـ هـىـ الإـشـارـةـ الصـحـيـحةـ لـلـنـتـيـجـةـ.ـ نـخـلـصـ مـنـ ذـلـكـ إـلـىـ النـتـيـجـةـ الـمـهـمـةـ التـالـيـةـ وـهـىـ أـنـ إـذـاـ تـمـ إـجـرـاءـ عـلـمـ حـسـابـيـةـ عـلـىـ

رـقـمـيـنـ وـكـانـ عـلـمـ الفـيـضـانـ يـسـاـوىـ وـاحـدـ،ـ فـإـنـ ذـلـكـ يـعـنـىـ أـنـ الإـشـارـةـ الصـحـيـحةـ لـلـنـتـيـجـةـ هـىـ المـوـجـودـةـ فـيـ عـلـمـ الإـشـارـةـ

S، وليست الموجودة في علم السالبة N. أما إذا كان علم الفيضان V يساوى صفر، فإن ذلك يعني أنه لا يوجد خطأ في إشارة النتيجة وستكون الإشارة الصحيحة هي الموجودة في كل من علم السالبة N أو علم الإشارة S حيث سيكونا متساوين. ربما يظهر سؤال هنا كيف تقوم وحدة الحساب والمنطق ALU بحساب قيمة علم الفيضان V. علم الفيضان يساوى ناتج عملية الإكس أور XOR على الحمل من الخانة السادسة إلى السابعة مع الحمل من الخانة السابعة إلى الخارج والذي يمثل علم الحمل. بتطبيق ذلك على المثال السابق ستتجدد أن الحمل من الخانة السادسة إلى السابعة يساوى 1، بينما الحمل من الخانة السابعة فيساوى صفر، وعلى ذلك فإن عملية الإكس أور على واحد وصفر تساوى واحد، وبالتالي كان علم الفيضان  $V=1$ . السؤال الثاني هنا هو، إذا كنا عرفنا أين نجد الإشارة الصحيحة في حالة أن علم الفيضان  $V=1$ ، فكيف نعرف القيمة الصحيحة للنتيجة؟ يمكننا أن نعرف النتيجة الصحيحة بإجراء الخطوتين التاليتين:

- ١ - في المثال السابق كانت النتيجة تساوى 10000000، إعكس بت الإشارة (بت الإشارة تساوى 1، يجعلها صفر) واحسب القيمة العشرية للناتج الجديد الذي يساوى 00000000، أى أن النتيجة بعد عكس بت الإشارة تساوى صفر.
- ٢ - إجمع 128 على هذه النتيجة الجديدة تعطيك الناتج الصحيح كما يلى  $128+128=256$ .

أنظر إلى المثال التالي لكي نؤكد على القواعد السابقة: أوجد حاصل جمع الرقم السالب  $-70 = 10111010$ ، مع نفسه. أى أننا سنجمع  $-70$  مع  $-70$  والمفروض أن يعطى  $140$ . أنظر ماذا يحدث:

$$\begin{array}{r} -70 \\ -70 + \\ \hline 01110100 \end{array}$$

هنا تم جمع رقمين كل منهما سالب (-70) فكانت النتيجة موجبة كما نرى حيث أن بت الإشارة (آخر بت) تساوى صفر، وهذا بالطبع خطأ كبير، والدليل على ذلك هو علم الفيضان V الذي يساوى 1 في هذه الحالة. لاحظ أن الحمل من الخانة السادسة للسابعة يساوى صفر، والحمل من الخانة السابعة يساوى واحد، وبالتالي فإن V الذي يساوى الإكس أور لهما يساوى 1. إذن طالما أن  $V=1$  فإن إشارة هذه النتيجة خطأ، والنتيجة نفسها خطأ أيضا. فكيف سنعرف الإشارة الصحيحة، والقيمة الصحيحة للنتيجة. الإشارة الصحيحة في هذه الحالة ستكون في العلم S الذي يساوى عملية الإكس أور على كل من علم الفيضان  $V=1$ ، وعلم السالبة (آخر بت في النتيجة)،  $N=0$ ، وبالتالي فإن  $S=1$ ، أى أن الإشارة ستكون سالبة. يبقى أن نعرف القيمة الصحيحة للنتيجة، وسيكون ذلك باتباع الخطوتين السابقتين كما يلى:

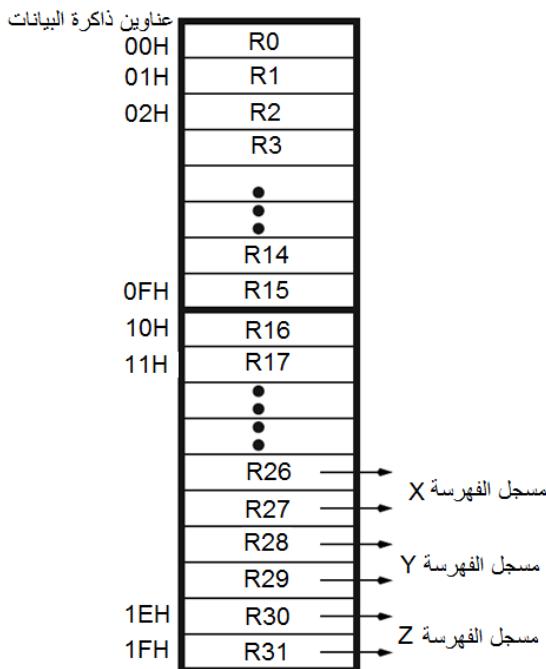
١- إعكس بت الإشارة في النتيجة السابقة وبالتالي ستصبح النتيجة الجديدة هي 11110100، وهذه النتيجة الجديدة سالبة ولكن نعرف قيمتها العشرية نحسب متممها الثنائي الذي يساوى 00001100، وبالتالي فإن النتيجة الجديدة تساوى 12- في النظام العشري.

٢- طالما أن النتيجة الجديدة سالبة سنجمع عليها 128- فتصبح النتيجة 140-128+ = (-12)، وبالتالي فإن

نتائج عملية الجمع ستكون 140-، وهي النتيجة الصحيحة.

إذن الخلاصة من ذلك هي أنه إذا كان علم الفيضان يساوى صفر فلا خطأ في النتيجة وإشارتها صحيحة، أما إذا كان علم الفيضان يساوى واحد، فإن إشارة النتيجة تكون في العلم S وليس العلم N، كما أن النتيجة تكون خطأ ويتم تصحيحها بالخطوتين السابقتين.

نعود الآن إلى استئناف الكلام عن محتويات المُتحكمات AVR.



شكل ٦-٣ المسجلات العامة في المُتحكمات AVR

### المسجلات العامة General purpose registers

عدد هذه المسجلات هو ٣٢ مسجلًا في كل إصدارات المُتحكمات AVR، وعدد بتاتها يساوى عدد برات مسار البيانات (٨ بت في المُتحكمات ٨ بت). هذه المسجلات يمكن التعامل معها بأسمائها بدءًا من R0 حتى R31، فمثلاً أمر التجميع MOV R3,R1 سيُنقل نسخة من محتويات المسجل R1 ويُضعها في المسجل R3. يمكن اعتبار هذه المسجلات جزءًا من ذاكرة البيانات الخاصة بالمُتحكم وعلى ذلك فإنه يمكن التعامل معها من خلال عنوان لكل منها حيث المسجل R0 يكون عنوانه هو 00H والمسجل R31 يكون عنوانه هو 1FH. الحرف H يعني أن الرقم المجاور مكتوب في النظام الثنائي. بعض هذه المسجلات يكون لها وظائف إضافية بجانب كونه أحد المسجلات العامة. فمثلاً المسجل R0 يستخدم مع أمر لغة التجميع لتحميل ذاكرة البرمجة، كما أن المسجلات R26 حتى R31 تستخدم كمسجلات فهرسة وهي أحد صور التعامل غير المباشر مع الذاكرة عند البرمجة بلغة التجميع. إننا في هذا الكتاب لن نركز على لغة التجميع كلغة برمجة للمُتحكم لأننا سنستخدم البرمجة بلغة C لأنها

هي الأشهر والأسهل والأكثر استخداماً هذه الأيام. شكل ٦-٣ يبين رسمياً توضيحاً للمسجلات العامة في المتحكمات AVR.

## الذاكرة Memory

هذه إما أن تكون ذاكرة برمجة أو ذاكرة بيانات وتختلف كميتها على حسب الإصدار المستخدم. كل من نوعي الذاكرة موصول على وحدة المعالجة المركزية CPU بطريقة هارفارد التي تم شرحها في الفصل السابق بحيث يتم الحفاظ على تنفيذ معظم أوامر المتحكم في دورة ساعة واحدة.

## ذاكرة البيانات Data memory

هذه الذاكرة من النوع المتطاير، بمعنى أن محتوياتها تفقد بانقطاع القدرة عن المتحكم. يمكن تقسيم هذه الذاكرة إلى خمسة أقسام في كل إصدارات المتحكمات AVR كما يلى:

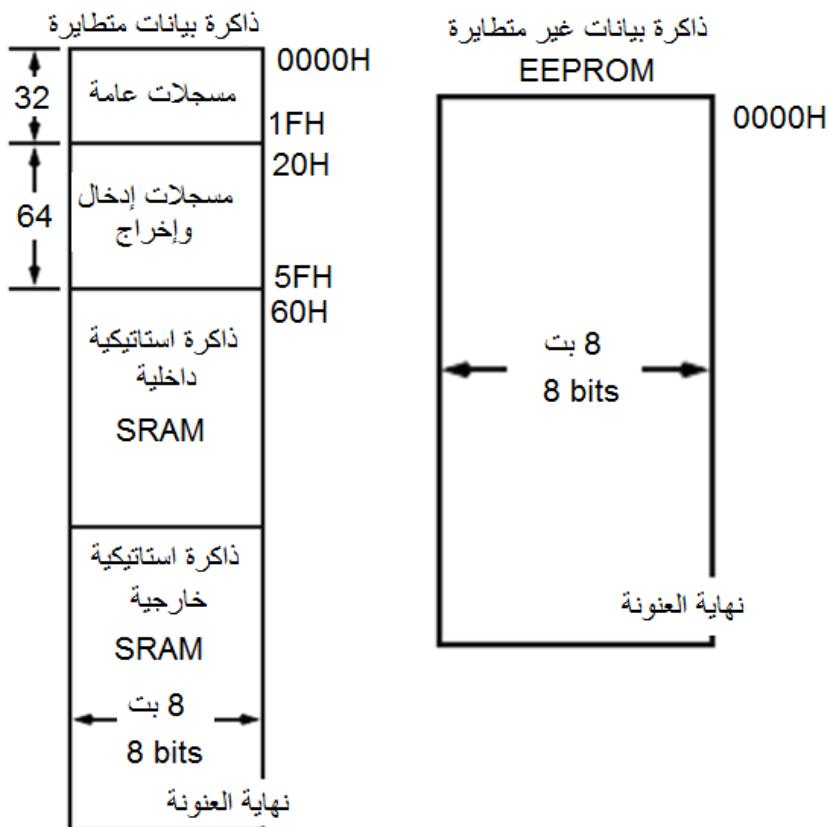
**١ - مجموعة المسجلات العامة المكونة من ٣٢ مسجلًا** والتي سبق شرحها في الجزء السابق، والتي تبدأ بالعنوان ٠٠H وتنتهي بالعنوان ١FH كما رأينا في شكل ٦-٣.

**٢ - هناك تقريراً ٦٤ مسجلًا تسمى مسجلات الإدخال والإخراج**، تزيد أو تقل على حسب إصدار المتحكم.

وهذه المسجلات خاصة بالتعامل مع الأجهزة الملحقة بالمحكم مثل المؤقتات، وبوابات إدخال وإخراج البيانات، والمتحول التماثلي والرقمي، وغير ذلك، وبالطبع فإن هذه الملحقات مختلف عددها باختلاف الإصدار. لذلك فإن هذه المسجلات تسمى مسجلات الإدخال والإخراج I/O registers حتى عنوان آخر مسجل في هذه المسجلات، أو يمكن التعامل معها بأسمائها الخاصة بما مثل المسجل DDRD وهو مسجل تحديد أطراف البوابة D في المحكم، أيها سيكون طرفاً لإدخال البيانات وأيها سيكون طرفاً لإخراج البيانات، والمسجل ADCSRA وهو مسجل الحالة والتحكم A، والذي يتم التحكم من خلال باته في أداء المتحول التماثلي الرقمي، وهكذا باقي المسجلات من هذا النوع.

**٣ - كمية من الذاكرة الاستاتيكية SRAM الداخلية**، تتراوح ما بين ١٢٨ بait حتى ٤ كيلوبايت على حسب إصدار المحكم. وهذه تبدأ عنوانينها بعد آخر عنوان في مسجلات الإدخال والإخراج السابقة، والوظيفة الأساسية لهذه الذاكرة هي استخدامها كمكدسة stack. عند النداء على برنامج فرعى، أو عندما يقفز المحكم إلى برنامج لخدمة مقاطعة معينة فإنه لابد من الاحتفاظ بعنوان الأمر الذى كان سينفذه المحكم قبل أن يقفز حتى يعود إليه مرة ثانية بعد أن ينتهى من تنفيذ برنامج خدمة المقاطعة أو

البرنامج الفرعى، وستتكلم عن ذلك بشيء من التفصيل عند الحديث عن المقاطعة فى هذه المتحكمات في فصل خاص بذلك. بالطبع يمكن استخدام هذه الذاكرة لتخزين أي بيانات مرحلية أثناء تنفيذ البرنامج. لاحظ أن هذه الذاكرة متطابقة بمعنى أنها تفقد محتواها بانقطاع القدرة مثل المسجلات السابقة. هذه الكمية من الذاكرة تكون موجودة داخل المتحكم حتى تميزها عن النوع التالى.



شكل ٧-٣ أقسام ذاكرة البيانات في المتحكمات AVR

**٤ - كمية من الذاكرة الاستاتيكية SRAM الخارجية**، وهذه يتم توصيلها عن طريق المستخدم من خارج المتحكم، وهذا بالطبع إذا كان إصدار المتحكم من النوع المؤهل لإمكانية توصيل هذه الذاكرة عليه من الخارج، ولذلك فإن هذه الإمكانية تكون موجودة فقط في الإصدارات ذات الإمكانيات العالية مثل المتحكم AT90S8515 وليس كل الإصدارات بالطبع بما هذه الإمكانية. تبدأ عناوين هذا الجزء من الذاكرة بعد آخر عنوان في الذاكرة الاستاتيكية الداخلية. شكل ٧-٣ يبين هذه الأجزاء الأربع من ذاكرة البيانات.

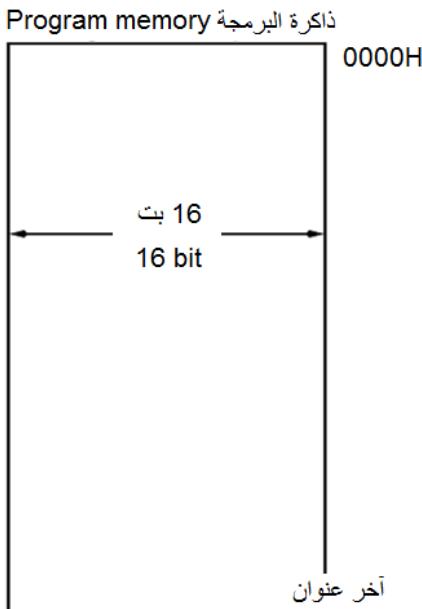
**٥ - كمية من الذاكرة غير المتطابقة EEPROM** تتراوح كميتها من ٦٤ بايت حتى ٤ كيلوبايت تستخدم في تخزين الثوابت التي يكون هناك خوف من فقدتها في حالة انقطاع القدرة. هذه الذاكرة لها مدى عنوان خاص بها، أي أن عناوينها ليست متتابعة بعد الذاكرة الاستاتيكية السابقة، ولكنها تبدأ من العنوان

0000H الخاص بها كما في شكل ٧-٣. يمكن التسجيل في هذه الذاكرة ومسحها حوالي ١٠٠ ألف مرة، ولا ينصح باستخدامها لتخزين البيانات المرحلية المتكررة لأن الكتابة القراءة منها تتم بطريقة خاصة وتأخذ وقتا طويلا، سترى ذلك في فصول قادمة.

### ذاكرة البرمجة Program memory

هذه الذاكرة عبارة كمية من الذاكرة اللحظية flash memory غير المنطابقة تستخدم في كتابة البرامج. تتراوح كمية هذه الذاكرة من ١ كيلوبايت (تقسم في صورة ٥١٢ بait × ١٦ بت) حتى ١٢٨ كيلوبايت (مقسمة في صورة ٤٦ كيلوبايت × ١٦ بت) على حسب إصدار المتحكم. كما تلاحظ فإن عرض البايت هنا يكون ١٦ بت، وهو أكبر عدد من البتات يمكن أن يشغلها أي أمر من الأوامر بحيث يتم جلب الأوامر في مشوار واحد فقط. دائما يوجد في بداية هذه الذاكرة مساحة تخصص لتخزين متوجه المقاطعة interrupt vector الخاص بالمتتحكم، ولذلك فإن كل البرامج يتم كتابتها بعد هذه المساحة المخصصة لمتوجه المقاطعة، كما سترى فيما بعد عند الحديث عن المقاطعة. متوجه المقاطعة يتغير أيضا على حسب إصدار المتحكم.

شكل ٨-٣ يبين توضيحا لهذا النوع من الذاكرة.



شكل ٨-٣ ذكرة البرمجة

### بوابات إدخال وإخراج البيانات

وهذه أيضا يختلف عددها باختلاف الإصدار الذي ستعامل معه، وستتكلم بالتفصيل عن ذلك مع الحديث عن المتحكم .atmega328

وحدة المقاطعة Interrupt unit وهذه سيتم إفراد فصل خاص بها في معرض الحديث عن المتحكم .atmega328

### وحدات ملحقة أخرى

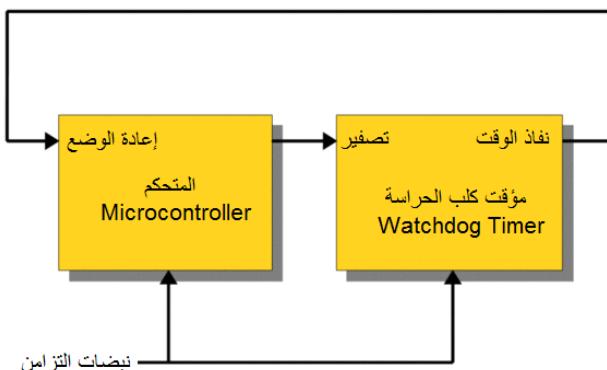
مثل المؤقتات، ومحول الإشارات التماضية إلى رقمية، ووحدة تحويل الإشارات من رقمي إلى تماضي، ووحدة الاتصالات المتزالية، وسيتم الحديث بالتفصيل عن كل واحد من هذه الوحدات في فصل خاص بكل منها وفي معرض الحديث

عن المتحكم atmega328 بالذات الذى سيصاحبنا في كل هذا الكتاب. لاحظ أن كل الوحدات الملحقة تتصل بمسار البيانات الخاص بالمتحكم كما في شكل ٣-٣.

### مؤقت كلب الحراسة Watchdog Timer

هذا المؤقت عبارة عن مؤقت من نوع خاص وله وظيفة خاصة جدا وهو موجود في معظم المتحكمات AVR. في بداية برنامجك تقوم بتفعيل هذا المؤقت مع ضبطه على زمن معين يكون مساوى تقريبا لزمن تنفيذ البرنامج أو أعلى منه قليلا. في نهاية البرنامج تضع أمرا يعلم على تصفيير المؤقت ومنعه من الفيضان. في أثناء تنفيذ البرنامج، إذا حدث أى شيء غير طبيعي (مثل دخول البرنامج في حلقة لا نهاية مثلا) فإن عملية التنفيذ لن تصل إلى أمر تصفيير المؤقت الذي يكون مستمرا في عملية العد التصاعدي، مما يتبع عنه فيضان للمؤقت، أى يتخطى الفترة الزمنية المحددة له. إذا حدث هذا الفيضان فإن المؤقت سيعيد وضع reset للمتحكم ويوقف عملية تنفيذ البرنامج

وبذلك يحمي البرنامج من أى أخطاء قد تتفاقم في عملية التنفيذ. معنى ذلك أن البرنامج إذا لم يقوم بتصفيير المؤقت (بمثابة طرد لكلب الحراسة)، فإن المؤقت (كلب الحراسة) سيفيض عن قيمته النهاية وسيقوم بناء على ذلك بإعادة وضع المتحكم منعا لأى تعقيبات أو مشاكل قد تحدث نتيجة الاستمرار في تنفيذ البرنامج. شكل ٩-٣ يبيّن رسميا توضيحا لهذه العملية، حيث بعد تنشيط كلب الحراسة يقوم المتحكم بتصفييره



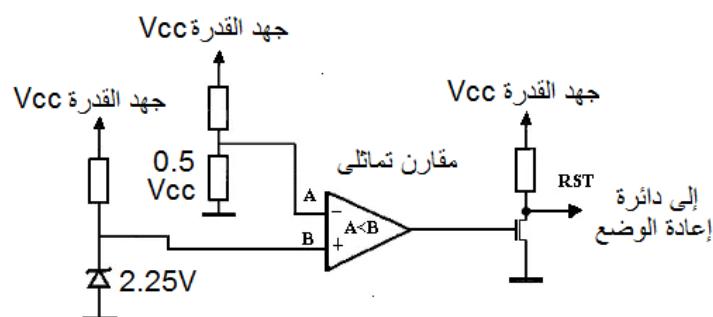
شكل ٩-٣ رسم صندوقى لمؤقت كلب الحراسة

باستمرار قبل أن يصل إلى نفاذ وقته ووصوله إلى الفيضان. كما نرى في الشكل إذا حدثت أى مشكلة في برنامج المتحكم منعه من أن يصفر المؤقت، فإن الوقت سينفذ، ويحدث فيضان للمؤقت، وسيقوم المؤقت بإعادة وضع (عمل reset) للمتحكم.

### دائرة متابعة مصدر القدرة Brownout detector

هذه الدائرة تكون موجودة أيضا في أغلب إصدارات المتحكمات AVR، بل في أغلب المتحكمات على وجه العموم. مهمة هذه الدائرة هي مراقبة جهد مصدر القدرة بحيث إذا نقص مصدر القدرة عن حد معين تقوم هذه الدائرة بإعادة وضع reset للمتحكم بطريقة آمنة بدلا من أن يتسبب الفيضان اللحظى في جهد مصدر القدرة في

حدوث فقدان أو تدمير لبعض البيانات المسجلة في المسجلات أو في ذاكرة البيانات. شكل ٣-١٠ يبين مثلاً على دائرة مقترحة لهذا الغرض. في هذه الدائرة يكون الدخل B للمقارن التماثلي عبارة عن جهد ثابت عند القيمة 2.25V ولن يتارجع لأنّه يمثل جهد الزيون دايوود (وذلك من خواص الزيون دايوود). الدخل A يأخذ نصف جهد القدرة (٥ فولت) من خلال مقسم الجهد بالمقاومتين الموضعتين في الشكل. بذلك يكون الدخل A للمقارن أعلى من الدخل B بمقدار ربع فولت، وفي هذه الحالة سيكون خرج المقارن يساوى صفر، وبالتالي سيكون ترانزستور الخرج غير موصل (OFF) وبالتالي فإنّ الجهد الواسط إلى دائرة إعادة الوضع سيساوى الجهد  $V_{CC}$ ، وستكون الدائرة مستقرة على هذا الوضع. إذا حدث نقص مفاجئ لجهد القدرة بحيث يجعل جهد الدخل A أقل من جهد الدخل B، فإنّ خرج المقارن سيصبح جهد عالي مما يجعل الترانزستور موصلًا (ON)، وبالتالي يصبح الجهد إلى دائرة إعادة الوضع يساوى صفر في هذا الحالة مما يتسبب في إثارة دائرة الوضع بحيث تقوم هي بإعادة وضع reset للمتحكم. من هذه الدائرة يمكنك أن تلاحظ أنّ التأرجح المسموح به في جهد القدرة يساوى 0.25 فولت. كل المعالجات والمتحكمات في الغالب تكون مجهزة بدائرة خاصة تقوم بإعادة وضع reset المعالج أو المتحكم بحيث تقوم هي بتصفير المسجلات أو وضع قيم ابتدائية مناسبة في كل منها بطريقة آمنة.



شكل ٣-١٠ دائرة مقترحة لمراقبة جهد القدرة

### ساعة الزمن الحقيقي Real Time Clock, RTC

ليست كل المعالجات أو المتحكمات مجهزة بهذه الساعة. هذه الساعة إن وجدت في أى متحكم فإنّها تعطيك إمكانية معرفة الزمن الحقيقي لأى حدث مثل الساعة واليوم والتاريخ، وعادة ما تكون هناك حاجة لذلك في الكثير من التطبيقات، حيث يمكن مثلاً طباعة زمن الإنتاج على أى منتج يتم إنتاجه.

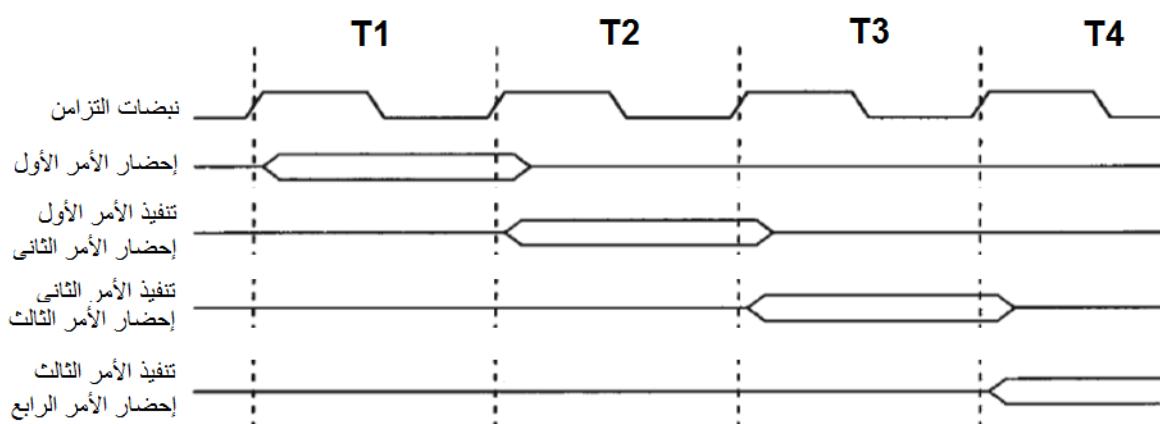
### ٣-٣ تفزيذ الأوامر في المتحكمات AVR

لقد رأينا في الفصل الثاني أن وحدة المعالجة المركزية CPU عندما تقوم بتنفيذ أي أمر فإنها تفعل ذلك على دورتين متتاليتين، الدورة الأولى هي دورة إحضار الأمر من الذاكرة ووضعه في مسجل الأوامر، وهذه الدورة تسمى عادة بدورة الإحضار fetching cycle، وأما الدورة الثانية والتي تبدأ بعد وصول شفرة الأمر إلى مسجل الأوامر فتسمى بدورة التنفيذ execution cycle. في هذا النظام، عندما تكون الوحدة مشغولة في دورة إحضار فإنها لا تفعل أي شيء آخر، وعندما تكون مشغولة في دورة التنفيذ فإنها لا تفعل أي شيء آخر، ولذلك فإن الزمن الكلى لتنفيذ أي أمر يكون مجموع زمني دوري الإحضار والتنفيذ معاً. أي أنه بفرض أن المعالج يحضر الأمر في نبضة تزامن واحدة، وينفذه في دورة تزامن واحدة (وهذا على سبيل المثال فقط)، فإن الزمن الكلى لتنفيذ الأمر في هذا المعالج سيكون اثنين من نبضات التزامن. هذا الوضع كان موجوداً في الأجيال الأولى من المعالجات (٨ بت)، ولكن تم استبداله فيما بعد بفكرة خطوط الإنتاج، أو خطوط الأنابيب pipelines.

### خطوط الأنابيب pipelines

T <sub>n</sub>	T <sub>n+1</sub>	T <sub>n+2</sub>
إحضار الأمر رقم n		
تنفيذ الأمر رقم n-1	إحضار الأمر رقم n+1	إحضار الأمر رقم n+2
	تنفيذ الأمر رقم n	تنفيذ الأمر رقم n+1

شكل ١١-٣ خط أنابيب المتحكم AVR المكون من مرحلتين، T<sub>n</sub> هي نبضة التزامن رقم n



شكل ١٢-٣ تتبع زمني لدخول الأوامر في خط الأنابيب بدءاً من الأمر الأول

المعالجات أو المتحكمات التي تعمل بهذا النظام ومنها المتحكمات AVR تقسم عملية تنفيذ أي أمر إلى مراحلتين منفصلتين أو متتاليتين، وكلا المراحلتين تعملان في نفس الوقت وباستمرار، والمراحلتان المتتاليتان هما مرحلة الإحضار ومرحلة التنفيذ. الجديد هنا أنه في أثناء انشغال وحدة الإحضار في إحضار الأمر رقم  $n$ ، فإن وحدة التنفيذ تكون مشغولة في نفس الوقت في تنفيذ الأمر السابق لهذا الأمر وهو الأمر  $n-1$ . بعد الانتهاء من ذلك، فإن الأمر رقم  $n$  ينتقل من مرحلة الإحضار إلى مرحلة التنفيذ ليبدأ في تنفيذه، ويدخل في نفس الوقت الأمر رقم  $n+1$  في مرحلة الإحضار، وهكذا. أي أنه عند أي لحظة، أو في أثناء أي نبضة تزامن تكون وحدة الإحضار مشغولة في تنفيذ أمر معين، ووحدة التنفيذ مشغولة في تنفيذ الأمر السابق له.

شكل ١١-٣ يبين رسمياً تخطيطياً لخط الأنابيب مكون من مراحلتين، حيث نلاحظ من هذا الشكل أنه عند أي نبضة تزامن تكون كل من المراحلتين مشغولتين في تنفيذ أمررين متتابعين، ولذلك فإن زمن التنفيذ الكلي لأي أمر من الأوامر سيكون نبضة تزامن واحدة فقط بدلاً من اثنين كما كان مسبقاً. شكل ١٢-٣ يبين التتابع الزمني لخط الأنابيب بدءاً من أول أمر حيث نلاحظ نفس الفكرة وبنفس الطريقة. فكرة خط الأنابيب مطبقة في الكثير من المعالجات والمتحكمات حتى أنها في المعالجات بنتيجة ٤ تصل إلى خمس مراحل بدلاً من مراحلتين كما في المتحكمات

.AVR

### ٣-٤ البناء المعماري RISC في مقابل البناء المعماري CISC

قبل أن نختتم هذا الفصل كان لابد من توضيح تعبيرين غایة في الأهمية ويستخدمان بكثرة جداً وبالذات عند ذكر مواصفات أي واحد من المتحكمات أو المعالجات. التعبير الأول هو CISC وهو اختصار للعبارة Complex Instruction Set Computer، والتي تعنى الحاسوبات ذات مجموعة الأوامر المركبة Complex. الكلمة مركبة هنا تعود على الأمر نفسه، أي أن الأمر يكون من النوع المركب، ولا تعود على مجموعة الأوامر. التعبير الثاني هو RISC وهو اختصار للعبارة Reduced Instruction Set Computer، والتي تعنى الحاسوبات ذات مجموعة الأوامر المبسطة أو المخفضة، وهي أيضاً هنا صفة للأمر نفسه، أي أن الأمر يكون مبسط، كما أن عدد الأوامر أيضاً يكون مخفضاً. ما هو الفرق بين هذين البنائيين المعماريين؟ وما هي مميزات وعيوب كل منهما؟

عند تصميم أي معالج أو متحكم يتم تحديد مجموعة الأوامر التي يمكن لهذا المعالج أو المتحكم أن ينفذها وهو ما يسمى بمجموعة الأوامر instruction set، وأيضاً يتم تحديد طرق التعامل مع الذاكرة من خلال هذه الأوامر وهو ما يسمى memory addressing modes. في العادة يتم قياس أداء المعالج أو المتحكم بقياس زمن تنفيذه لأى برنامج. زمن تنفيذ أي برنامج يمكن وضعه في المعادلة التالية:

زمن تنفيذ البرنامج = عدد الأوامر في البرنامج × عدد نبضات تنفيذ كل أمر × زمن كل نبضة

$$(1) \quad \text{عدد الأوامر}/\text{البرنامج} \times \text{عدد النبضات}/\text{الأمر} \times \text{الزمن}/\text{النبضة}$$

النبضة هنا تعنى بها نبضات التزامن أو دورة الماكينة (ودورة الماكينة تتكون من عدد من نبضات التزامن وتختلف باختلاف المعالج أو المتحكم). سنفترض أن زمن كل نبضة (الزمن/النبضة) ثابت. لذلك بالنظر إلى المعادلة (1) سنجد أنه يمكن تقليل زمن تنفيذ البرنامج (تحسين الأداء) بطريقتين مختلفتين: الطريقة الأولى عن طريق تقليل عدد الأوامر في البرنامج، والطريقة الثانية عن طريق تقليل عدد النبضات لكل أمر، ومن هنا نشأت المدرستان التاليتان:

- المدرسة الأولى هي مدرسة ال CISC: وهم الذين اختاروا تقليل زمن تنفيذ البرنامج عن طريق تقليل عدد الأوامر في البرنامج عن طريق جعل الأوامر أكثر تعقيداً، بصرف النظر عن عدد النبضات لكل أمر. أي أن الأمر يؤدي أكثر من عملية أولية مثل إحضار الأمر، وإحضار المعاملات، ثم إجراء العملية المطلوبة من هذا الأمر، ثم تخزين النتيجة. كل ذلك يتم في أمر واحد فقط، لذلك كان الأمر مركباً complex.

لذلك فإن أزمنة تنفيذ كل أمر لن تكون متساوية ولكنها ستختلف باختلاف المهمة التي يؤديها هذا الأمر، وبالطبع سيكون عدد الأوامر المطلوبة أكثر، لأننا سنحتاج أمر لكل مهمة مركبة.

- المدرسة الثانية هي مدرسة ال RISC: وهم الذين اختاروا تقليل زمن تنفيذ البرنامج عن طريق تقليل عدد النبضات لكل أمر مع التضحية بعدد الأوامر في البرنامج. لذلك فإن كل أمر هذا النوع من الهيكلة أو البناء العمارات تكون من النوع الأولي، أي أنها تؤدي مهمة أولية واحدة فقط لا غير، مثل جمع محتويات مسجلين، أو نقل محتويات مسجل إلى الذاكرة أو العكس، وهكذا. لذلك فإن كل هذه الأوامر تكون لها أزمنة تنفيذ متساوية، وهذه ميزة مهمة في هذا النوع، كما أن كلها تنفذ تقريباً في دورة (أو نبضة واحدة). كما أن عدد هذه الأوامر سيكون أقل لأنه محدود بالعمليات الأولية فقط. لكن فهم الفرق بين المدرستين ستفترض المثال التالي: المطلوب حساب حاصل ضرب الرقم الموجود في عنوان الذاكرة a في الرقم الوجود في عنوان الذاكرة b ووضع الناتج في العنوان a.

عند تنفيذ هذه المهمة (ضرب الرقمين السابقين) باستخدام الطريقة الأولى CISC، في هذه الحالة سيكون البرنامج عبارة عن أمر واحد فقط وهو:

MUL a,b

حيث MUL هي اختصار لكلمة multiplication، وهنا سيقوم الأمر بإحضار المعامل الأول من العنوان a في الذاكرة، ثم المعامل الثاني من العنوان b، ثم ضربهما، ثم تخزين النتيجة في العنوان a. أمر واحد مركب قام بتنفيذ الكثير من العمليات الأولية.

أما عند تنفيذ هذه المهمة (ضرب الرقمين) باستخدام الطريقة الثانية RISC، في هذه الحالة سيكون البرنامج مكون من عدة أوامر أولية كما يلى، مع العلم أنه في هذه الطريقة لابد أن يحتوى المعالج أو المحكم على مجموعة من المسجلات العامة A و B و C و D مثلا:

Load A,a

Load B,b

MULT A,B

Store a,A

كما تلاحظ أنه في هذه المرة تم تنفيذ البرنامج في أربعة أوامر، ولكنها كلها أوامر أولية، حيث الأمر الأول يحضر محتويات عنوان الذاكرة a إلى المسجل A، والأمر الثاني يحضر محتويات عنوان الذاكرة من العنوان b إلى المسجل B، والأمر الثالث يجرى عملية حسابية (MULT) على محتويات مسجلين (A و B) ويوضع الناتج في المسجل A، وأما الأمر الثالث فإنه يخزن محتويات المسجل A في العنوان a في الذاكرة. نلاحظ هنا أن جميع الأوامر من النوع الأولى (يؤدى مهمة واحدة فقط)، كما أن جميع الأوامر تحتاج لنفس الزمن لتنفيذ كل منها، والملاحظة الثالثة أن هذه الطريقة احتاجت إلى مسجلات عامة وذلك لأن أي عملية حسابية تجرى على مسجلات فقط. من هنا يمكننا أن نكتب الميزات التالية لكل واحد من هذين الهيكلين أو البنائيين المعماريين:

#### النظام CISC المعماري:

- أوامر مركبة تحتاج للعديد من النبضات من أجل التنفيذ
- تشتمل الأوامر على عمليات التخزين والاستدعاء من الذاكرة
- عدد الأوامر في البرنامج يكون أقل، ولكن كل أمر يحتاج للعديد من نبضات الساعة
- الأوامر تنفذ في أزمنة مختلفة اعتمادا على نوع الأمر
- من الصعب تنفيذها باستخدام طريقة خطوط الأنابيب pipelines من أجل إسراعها.
- لا تحتاج للكثير من المسجلات العامة.

#### النظام RISC المعماري:

- أوامر أولية يحتاج كل منها في الغالب لنبضة تزامن واحدة للتنفيذ
- التخزين والاستدعاء من الذاكرة يكون بأمررين منفصلين
- عدد الأوامر في البرنامج يكون أكثر، ولكن كل أمر يحتاج لنبضة ساعة واحدة للتنفيذ
- جميع الأوامر تقريبا تنفذ في أزمنة مت Rowe

- حيث أن أزمنة تنفيذ الأوامر تكون متساوية فإنه يكون من السهل تنفيذها باستخدام طريقة خطوط الأنابيب من أجل إسراعها.
- تحتاج للكثير من المسجلات العامة أو ما يسمى ملف المسجلات الذي يحتوى على الأقل على أكثر من ٣٠ مسجلًا.

معظم المتحكمات ومنها المتحكمات AVR تقوم على العمارة بطريقة ال RISC ولقد رأينا ذلك في استخدام خطوط الأنابيب pipelines عند تنفيذ الأوامر، كما رأينا أن ملف المسجلات يحتوى على ٣٢ مسجلًا، وستجد أن هذا المعنى يتم التركيز عليه في الكالوج أو جداول البيانات الخاصة بأى متحكم أو معالج. بالنسبة وكما رأينا السبب في تسمية لوحات الأردوينو بهذا الإسم في بداية هذا الفصل، فما هو السبب في تسمية المتحكمات AVR بهذه المسمى؟ الأمر مشابه تماماً لسبب تسمية الأردوينو حيث هنا كان الطالبان Alf-Egil Bogen و Vegard Wollan طالبين في معهد النرويج للتكنولوجيا، وقاما بتصميم مشروع متحكم يعمل بنظام RISC المعماري لما له من مميزات ذكرناها مسبقاً وأسموه AVRisc حيث AV هما أول حرفين من أسمائهم كما نرى، ومن هنا كانت التسمية AVR.

## ٥-٥ تداول البيانات داخل المتحكمات AVR

أثناء إجراء أي عملية تقوم بها وحدة الحساب والمنطق ALU كيف يتم التعامل مع المعاملات، وأين تخزن النتيجة؟ هناك عدة طرق لهذا التعامل ومنها ما يلى:

### ١- حواسيب المكذسة stack machine

في هذه المعالجات أو المتحكمات لا يكون تعامل لوحة الحساب والمنطق ALU إلا مع المكذسة stack، حيث تقوم وحدة الحساب والمنطق بسحب المعاملات من قمة المكذسة، ثم تقوم بتنفيذ عملية الحساب، ويتم تخزين النتيجة في المكذسة مرة ثانية. هناك أمر التخزين في المكذسة وهو الأمر PUSH، وأمر السحب من المكذسة وهو الأمر POP. لبيان ذلك افترض أننا نريد إجراء عملية الطرح التالية:  $C = A - B$ . البرنامج الذي سيقوم بهذه العملية يمكن كتابته كما يلى:

```
PUSH A
PUSH B
SUB
POP C
```

حيث الأمر الأول يدفع بالمتغير A في قمة المكدة، ثم الأمر الثاني يدفع بالمتغير B في قمة المكدة مع الإزاحة بالمتغير A إلى داخل المكدة بمقدار موضع آخر، ثم الأمر الثالث (SUB) يقوم بطرح آخر مكانين في قمة المكدة (A-B) ويضع الناتج في قمة المكدة أيضاً، وهكذا أى عملية تقوم وحدة الحساب والمنطق بإجرائها. الأمر الأخير يسحب محتويات قمة المكدة ويضعها في المتغير C. هذا النوع من التعامل كان موجوداً في الآلات الحاسبة المبرمجة ولكنه لا يستخدم الآن في أى من المعالجات أو المتحكمات.

## ٢- حواسيب المركم accumulator machines

هنا تتم كل العمليات الحسابية والمنطقية من خلال المركم، حيث لابد أن يكون أحد المعاملات في المركم، وأما المعامل الآخر فيكون في أى مكان آخر (مثل الذاكرة أو أحد المسجلات الأخرى)، وبعد إجراء العملية توضع النتيجة في المركم أيضاً. حيث سيكون البرنامج السابق لإجراء عملية الطرح كما يلى:

Load A

Sub B

Store C

حيث الأمر الأول يضع المعامل أو الرقم الأول A في المركم، والأمر الثاني يطرح الرقم الثاني B من المركم ويضع النتيجة في المركم أيضاً، والأمر الثالث يخزن محتويات المركم في المتغير C. كل المعالجات ٨ بت (أول جيل من المعالجات) كانت تعمل بهذه الطريقة. لاحظ أن A و B و C من الممكن أن تشير إلى عناوين في الذاكرة.

## ٣- حواسيب الذاكرة والمسجل register-memory machines

هي نفسها تقريباً الحواسيب المعتمدة على المركم، ولكن هنا يمكن للمعاملات أن تكون في أى مسجل وليس بالضرورة أن تكون في المركم كما سبق وسيكون البرنامج السابق كما يلى:

Load Rx, A

Sub Rx, B

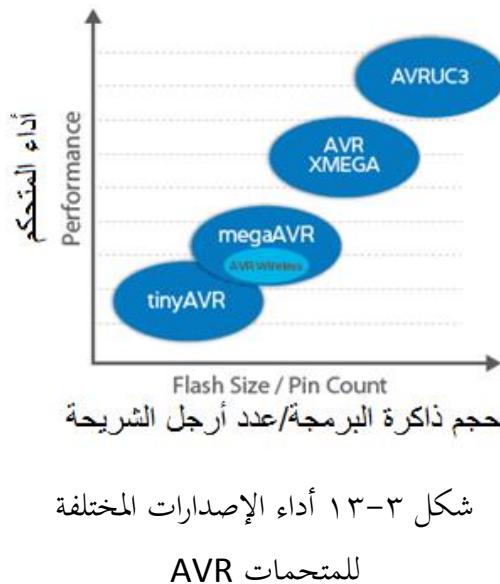
Store C, Rx

الأمر الأول يضع المتغير A في المسجل Rx، والأمر الثاني يطرح المتغير B من المسجل Rx ويضع الناتج في المسجل Rx، والأمر الثالث يخزن محتويات المسجل Rx في المتغير C. المعاملات A و B و C من الممكن أن تشير إلى عناوين في الذاكرة.

## ٤- حواسيب المسجل للمسجل register-register machines

هنا لابد أن تكون كل المعاملات في المسجلات، وسيتم تنفيذ البرنامج السابق كما يلى:

Load Rx, A  
Load Ry, B  
Sub Ry, Rx  
Store C, Ry



الأمر الأول يضع المتغير A في أحد المسجلات Rx، والأمر الثاني يضع المتغير B في مسجل آخر Ry، والأمر الثالث يطرح المسجل Rx من المسجل Ry ويضع النتيجة في المسجل Ry، وأما الأمر الرابع فيضع محتويات المسجل في المتغير C. لاحظ هنا أن العملية الحسابية تتم على محتويات مسجلات فقط، ولا يمكن أن تتم مع ذاكرة. المتحكمات AVR تقوم على هذا النوع من التعامل مع البيانات، وكما قلنا من قبل أن هذا النوع يحتاج للكثير من المسجلات العامة، أو ملف مسجلات register file كبير.

## ٦-٣ نظرة موسعة على منتجات أقل من المتحكمات AVR

تنبع شركة أتمل وهى من أشهر شركات إنتاج الإلكترونيات سلسلة عريضة من إصدارات المتحكمات AVR ذات الإمكانيات المختلفة التي تناسب جميع التطبيقات تقريباً بحيث يمكنك اختيار المتحكم المناسب للتطبيق بالضبط ولا تكون مضطراً لاستخدام متحكمات ذات إمكانيات عالية في تطبيقات بسيطة بحيث يكون هناك إسراف وإضاعة لأموال في إمكانيات لا يتم الاستفادة منها. شكل ١٣-٣ يبين رسمياً تخطيطياً لتغيير أداء هذه العائلة من المنتجات مع تغيير كمية ذاكرة البرمجة وعدد أطراف الشرائح في كل فصيل. نلاحظ في هذا الشكل تقسيم المنتجات إلى أربع فصائل أبسطها من حيث الإمكانيات هي الإصدارات tinyAVR، وأعلاها هي الإصدارات AVRUC3 وسنلقى نظرة سريعة على إمكانيات كل فصيل من الفصائل الموضحة في شكل ١٣-٣.

### ١- مجموعة المتحكمات ATtinyAVR

تتميز هذه المجموعة بأنها الأقل من حيث حجم ذاكرة البرمجة وعدد أطراف إدخال وإخراج البيانات. تحتوى هذه المجموعة على حوالي ٤٠ إصدار (شريحة AVR)، أقلها من حيث الإمكانيات هي الشرائح AVR4 و ATtinyAVR4 و ATtinyAVR9 و ATtinyAVR5 و ATtinyAVR10 التي لها ٤ أطراف فقط لإدخال أو إخراج البيانات،

وواحد كيلوبايت أو نصف كيلوبايت من ذاكرة البرمجة، و ٣٢ بait فقط من الذاكرة العشوائية (الاستاتيكية) وهى ملف المسجلات فقط، بعضها يحتوى على عدد واحد فقط، وبعضها يحتوى محول تماثلى رقمي ٨ بت، وبعضها يحتوى على قنوات تعديل عرض الموجة PWM الذى يمكن استخدامه فى التحكم في سرعة المواتير، وكلها تعمل على نبضات تزامن تصل إلى ١٢ ميجا هرتز. أعلى هذه المجموعة من حيث الإمكانيات هو المتحكم ATtiny1634 الذى به ١٨ طرف لإدخال وإخراج البيانات، و ١٦ كيلوبايت من ذاكرة البرمجة (فلاش)، وواحد كيلوبايت من الذاكرة العشوائية الاستاتيكية (RAM)، ومؤقت ٨ بت وآخر ١٦ بت، ومحول تماثلى رقمي ١٢ بت، وقناتين من قنوات ال PWM. للحصول على تفاصيل أكثر عن هذه المجموعة من المتحكمات AVR يمكن الرجوع إلى الموقع [https://en.wikipedia.org/wiki/Atmel AVR\\_ATtiny\\_comparison\\_chart](https://en.wikipedia.org/wiki/Atmel AVR_ATtiny_comparison_chart)

جدولاً يبين إمكانيات كل شرائح هذه المجموعة ومراجع للبيانات التفصيلية لكل منها.

## ٢- مجموعة المتحكمات ATmegaAVR

تحتوى هذه المجموعة على حوالي ٤١ إصداراً أو شريحة كلها من النوع ٨ بت، تتراوح كمية ذاكرة البرمجة لهذه المجموعة من ٤ كيلوبايت حتى ١٢٨ كيلوبايت، وعدد أطراف إدخال وإخراج البيانات يتراوح من ٢٣ حتى ٥٣ طرفاً. كمية الذاكرة العشوائية SRAM في هذه الشرائح تتراوح من نصف كيلوبايت حتى ١٦ كيلوبايت، وكل هذه الشرائح تحتوى على محول تماثلى رقمي ١٠ بت تتراوح قنواتها التماثلية ما بين ٦ و ٨ قنوات. كل هذه الشرائح تحتوى على ما بين ٣ إلى ٦ قنوات تعديل عرض الموجة PWM، وتتراوح نبضات التزامن لهذه المجموعة ما بين ٨ و ٢٠ ميجا هرتز، وكلها بها إمكانية البيانات التتابعية. للحصول على تفاصيل أكثر عن هذه المجموعة من المتحكمات AVR يمكن الرجوع إلى الموقع:

[http://www.futurlec.com/ICAtmel\\_ATMega\\_Comparison.shtml](http://www.futurlec.com/ICAtmel_ATMega_Comparison.shtml)  
كل شرائح هذه المجموعة مع إمكانية تنزيل كتالوج data sheet لكل واحدة من هذه الشرائح.

## ٣- مجموعة المتحكمات AVR XMEGA

تحتوى هذه المجموعة على عدد من المتحكمات AVR ذات ٨ أو ١٦ بت عالية الإمكانات، وهذه المجموعة مقسمة إلى أربع مجموعات جانبية كل منها تحتوى العديد من الشرائح ذات الإمكانات المختلفة. عموماً تتراوح أرجل هذه الشرائح من ٤٤ حتى ١٠٠ طرف منها ٣٤ حتى ٧٨ طرفاً لإدخال وإخراج البيانات. تتراوح ذاكرة البرمجة في هذه الشرائح من ١٦ حتى ٢٥٦ كيلوبايت، والذاكرة العشوائية من ٢ حتى ١٦ كيلوبايت. تحتوى على العديد من المؤقتات ذات ١٦ بت و ٨ بت. تحتوى على محولات تماثلية رقمية ١٢ بت بمعدل أخذ عينات أعلى بكثير من المتحكمات السابقة وعدد من القنوات ما بين ٨ و ١٢ قناة للدخل التماثلى. كل

شائحة هذه المجموعة تحتوى على محولات رقمية تماثيلية DAC ذات 12 بت وكلها لها قناتين للخرج التماثلى، وهذه ميزة أيضا لم تكن موجودة في كل المتحكمات السابقة. هذا بالإضافة إلى الكثير من الإمكانيات العالية الأخرى الغير موجودة في المتحكمات السابقة ويعنى من يحتاج التعامل مع هذه المتحكمات الحصول على بياناتها إما من موقع شركة أتمل نفسها أو الموقع <http://www.atmel.com/images/doc8077.pdf> الذى يحتوى شرحًا مفصلاً لمحطيات هذه المجموعة من الشرائح.

#### ٤- مجموعة المتحكمات AVRUC3

هذه المجموعة مقسمة أيضًا إلى عدد من المجموعات الجانبيه وكلها من النوع ٣٢ بت وتحتوى الكثير من الإمكانيات العالية التي تميزها عن المجموعتين الأول والثانية بالذات. تحتوى كميات كبيرة من ذاكرة البرمجة والذاكرة العشوائية وتتراوح أرجلها من ٤٨ حتى ١٤٤ طرف، وبالتالي الكثير من أطراف إدخال وإخراج البيانات. نبضات الساعة لهذه المتحكمات تصل إلى ٦٦ ميجاهرتز. يمكن الدخول على موقع أتمل والحصول على الكثير من الإمكانيات الإضافية لهذا الإصدار من الشرائح، والتي لا نريد الإطالة في تفاصيلها حيث أن هذا الكتاب ليس مجالاً للشرح التفصيلي لها.

### ملخص الفصل

ركز هذا الفصل على التركيب أو الهيكل (architecture) الداخلي للمتحكمات AVR على وجه العموم. من خلال ذلك تم استعراض ملف المسجلات وأنواع الذاكرة في هذه المتحكمات مع شرح مفصل لمسجل الحالة ووظيفة كل واحد من الأعلام الموجودة فيه. تم أيضًا شرح طريقة خطوط الإنتاج أو الأنابيب pipelining لتنفيذ الأوامر، والفرق بين الأجهزة CISC و RISC. ثم ختم الفصل بنظرة موسعة على جميع منتجات شركة أتمل والفرق في إمكانيات كل منها.

## الفصل ٤

### إدخال وإخراج البيانات في المتحكم atmega328

### Data I/O in The Atmega328 Microcontroller

**العناوين المضيئة في هذا الفصل:**

- ١- خصائص المتحكم atmega328
- ٢- إدخال وإخراج البيانات
- ٣- وسط البرمجة أتمل استديو
- ٤- برنامج محاكاة الدوائر الإلكترونية بروتس
- ٥- حرق البرنامج على شريحة المتحكم
- ٦- إنارة دايدود ضوئي وإطفاؤه بمعدل معين
- ٧- التعامل مع بثبات محددة في أي متغير

## ٤-١ مقدمة

بعد أن أخذنا فكرة عن المتحكمات على وجه العموم، ثم ركزنا على المعالجات AVR في الفصول السابقة، فإننا سنركز الحديث في هذا الفصل وما يليه على إصدار أو شريحة معينة من شرائح المتحكمات AVR وهو المتحكم atmega328، أما لماذا هذا الإصدار بالذات فقد أشرنا لذلك من قبل وهو أن لوحات الأردوينو تستخدم هذا المتحكم في تصميمها، ويمكن للقارئ أن يتبع ما يتم شرحه هنا على أي شريحة يريدها بالتوالي حيث ستكون الفروق بسيطة جدا. برمجة أي متحكم يمكن أن تكون باستخدام لغة التجميع الخاصة بهذا المتحكم، أو باستخدام أحد اللغات ذات المستوى العالي مثل لغة C++، والتي سنتخدمها هنا لسهولة التعامل بها عن استخدام لغة التجميع. لذلك



شكل ٤-٤ شكل توضيحي لمبرمج المتحكم وهو في التطبيق

المختلفة وكيف يتغير كل منها مع تنفيذ البرنامج، مما سيكون له ميزة كبيرة في تتبع تنفيذ البرنامج واكتشاف أخطاؤه. لذلك فإن تطوير أي تطبيق باستخدام أحد المتحكمات AVR سيجعلنا بحاجة إلى:

- ١- حاسب آلي بإمكانيات عادية
- ٢- برنامج أقل استديو أو أي برنامج محاكاة آخر لمن يفضل استخدام هذه البرامج لأنه قد تعود عليها من قبل. بعد التأكد من صحة البرنامج بلغة C وخلوته من أي أخطاء، وتنفيذه على المحاكى لرؤيه خرجه، يتم تحويل البرنامج إلى شفرات ثنائية أو ستعشرية مناسبة لتخزينها في ذاكرة البرمجة الخاصة بالمتحكم. هذه الصورة الثنائية للبرنامج يتم نقلها من الحاسب الآلى (من برنامج أقل استديو) إلى المتحكم عن طريق دائرة تسمى دائرة برمجة المتحكم، أو حرق البرنامج

كما يقال أحياناً، وهو في التطبيق In System Programmer, ISP، يتم توصيلها بين الحاسب الآلي والمتحكم المثبت في مكانه في التطبيق. هذه الدائرة يمكن شراؤها جاهزة من محلات الإلكترونيات، أو يمكنك أن تقوم بتجديدها بنفسك حيث توجد الكثير من مثل هذه الدوائر على الإنترنت. شكل ٤-١ يبين رسمياً توضيحاً لأحد هذه الدوائر حيث توصل من ناحية على الحاسب الآلي من خلال الـ USB أو المخرج المتوازي، ومن الناحية الأخرى توصل على المتحكم. يمكنك استخدام لوحة الأردوينو لنقل البرنامج على شريحة المتحكم أيضاً لمن يملكون أحد لوائح الأردوينو. على ذلك فإن المكون الثالث الذي سنكون بحاجة إليه سيكون هو دائرة برمجة أو حرق البرنامج على المتحكم.

هناك الكثير من دوائر المحاكاة للدوائر الإلكترونية ومن أشهرها برنامج بروتيس Protues الذي يمكنك من بناء أي دائرة إلكترونية في وسط التحرير الخاص به ثم توصل الدخل على هذه الدائرة وتنفذ البرنامج حيث يمكنك رؤية استجابة الدائرة أو خرجها لهذا الدخل. يحتوي برنامج البروتيس على كثير من وسائل العرض التي تمكنك من رؤية خرج الدائرة، ومنها مثلاً أوسولوسكوب، أو لمبات البيان LEDs، أو شاشات العرض LCDs، أو حتى موافير يمكنك استخدامها في حالة أن خرج الدائرة يغذى موتور حيث في هذه الحالة يمكنك توصيل المotor على خرج الدائرة ورؤيته وهو يدور بناء على خرج الدائرة. بناء على ذلك فإننا في هذا الكتاب سنتبع الخطوات التالية لإثبات صحة جميع البرامج التي سنستخدمها في كل أمثلة هذا الكتاب وكل التطبيقات التي سنقوم بتنفيذها فيه:

- ١ - سنقوم ببناء الدائرة أو التطبيق بالكامل على برنامج البروتيس وبالتالي سيكون أحد مكوناته هي شريحة المتحكم الذي نتعامل معه.
- ٢ - سنكتب البرنامج على برنامج أتمل استديو ونتأكد من صحة تشغيله على المحاكى المصاحب لهذا البرنامج.
- ٣ - نحصل على النسخة الثانية أو الستعشرية للبرنامج على أحد الملفات.
- ٤ - نضع هذه النسخة الثانية أو الستعشرية من البرنامج في ذاكرة البرمجة في شريحة المتحكم الموجودة في الدائرة المبنية في برنامج البروتيس.
- ٥ - نقوم بتنفيذ المحاكاة للدائرة أو التطبيق في البروتيس. إذا كان كل شيء على ما يرام ستزداد نتيجة التنفيذ على خرج الدائرة، وإذا كانت هناك أي أخطاء (إما في البرنامج أو في توصيل التطبيق على البروتيس)، فإنك ستعيد الكوة وتحاول تصحيح هذه الأخطاء، وتعيد المحاكاة إلى أن تتأكد من صحة البرنامج.
- ٦ - إذا تمت الخطوة ٥ بنجاح يمكنك بناء الدائرة الفعلية مستخدماً المكونات الحقيقية ونقل البرنامج على شريحة المتحكم الفعلية عبر دائرة برمجة المتحكم الموضحة في شكل ٤-١ وتشغيل الدائرة أو التطبيق في الصورة النهائية. بالطبع فإن كل من برنامج أتمل استديو والبروتيس يحتاج كل منهما لفصل منفصل لشرح كيفية استخدام كل منهما، ولكننا توفيراً للمكان فإننا سنعتبر أن القارئ لديه فكرة عن برنامج بروتيس وإذا لم يكن ذلك فيمكن الحصول على

ذلك من الإنترت المليئة بالمادة العلمية بصورها المختلفة التي تبين كيفية استخدام برنامج البروتوكول. أما برنامج أتمل استديو فإننا سنقوم بشرحه جزئيا وبالتابع مع الأمثلة والمشاريع التي سنسوقها في هذا الكتاب.

## ٤- ٢ خصائص المتحكم atmega328

سنسوق في هذا الجزء خصائص شريحة المتحكم atmega328 باختصار لأننا كما سنرى أن كل هذه الخصائص تقريباً متوافقة مع الخصائص العامة للمتحكمات AVR التي شرحناها في الفصل السابق. من هذه الخصائص ما يلى:

١- الشريحة مجهزة بعدد ٣٢ من المسجلات العامة المربوطة بوحدة المعالجة المركزية CPU وكلها مسجلات ٨ بت، كما أن هذا المتحكم يتبع تصميم هارفارد لتوصيل ذاكرة البرمجة والبيانات على وحدة المعالجة المركزية.

٢- هذا المتحكم مثل جميع المتحكمات AVR يتبع الحاسوبات ذات مجموعة الأوامر المبسطة RISC والتي تم شرحها في الفصل السابق، وعدد هذه الأوامر هو ١٣١ فقط من أوامر التجميع الأولية أو البسيطة. كما أنه يتبع نظام المسجل - للمسجل في تداول البيانات والذي شرحناه في الفصل السابق أيضاً والذي يفترض التعامل مع البيانات الموجودة في المسجلات العامة فقط، وكل هذه الأوامر تنفذ في دورة أمر واحدة نتيجة خط الأنابيب المكون من مرحلتين. هذا المتحكم يمكنه إجراء عمليات الضرب، ولكنها حالة خاصة تنفذ في دوري أمر على العكس من معظم الأوامر الأخرى.

٣- هذه الشريحة مزودة بذاكرة برمجة غير متطرافية ( فلاش ) مقدارها ٣٢ كيلوبايت، ويمكن برمجة المتحكم وهو في التطبيق.

٤- بها ١ كيلوبايت من ذاكرة البيانات غير المتطرافية EEPROM لتسجيل البيانات التي يخشى من ضياعها بانقطاع القدرة.

٥- ذاكرة عشوائية داخلية SRAM متطرافية مقدارها ٢ كيلوبايت.

٦- الشريحة بها اثنين مؤقت ٨ بت بالإضافة إلى مؤقت واحد ٦ بت، سيتم دراستها وبرجمتها بالتفصيل.

٧- الشريحة بها ساعة للزمن الحقيقي بمذبذب خاص بها للحصول على الوقت والتاريخ.

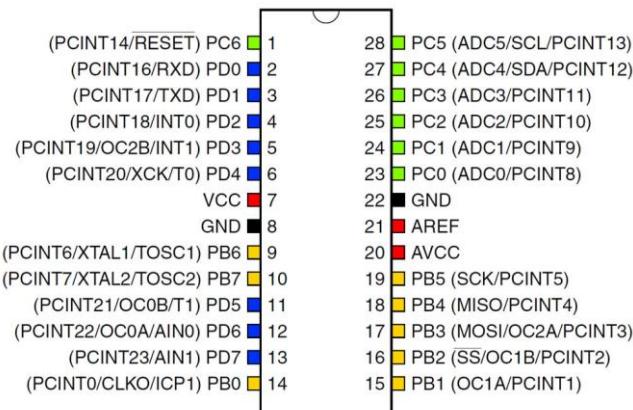
٨- ستة قنوات لتعديل عرض الموجة PWM والتي سيتم شرحها واستخدامها بالتفصيل.

٩- محول تماشى رقمي ADC ٨ بت مزود بستة قنوات دخل تماشى.

١٠- إمكانية التعامل مع البيانات التتابعية من خلال USART وسيتم شرح ذلك بالتفصيل أيضاً.

١١- مؤقت كلب حراسة.

- ١٢ - خطين للمقاطعة الخارجية، وإمكانية المقاطعة عند حدوث أي تغيير، بالإضافة إلى العديد من مصادر المقاطعة الأخرى، وسنجري ذلك بالتفصيل أيضا.
- ١٣ - دائرة مراقبة مصدر القدرة.
- ١٤ - إمكانية التعامل مع نبضات تزامن داخلية أو خارجية.
- ١٥ - عدد ٢٣ من خطوط إدخال وإخراج البيانات.
- ١٦ - تعمل على مصدر قدرة من ١.٨V حتى ٥.٥V.
- ١٧ - نبضات ساعة بسرعة تصل إلى ٢٠ ميجاهرتز.

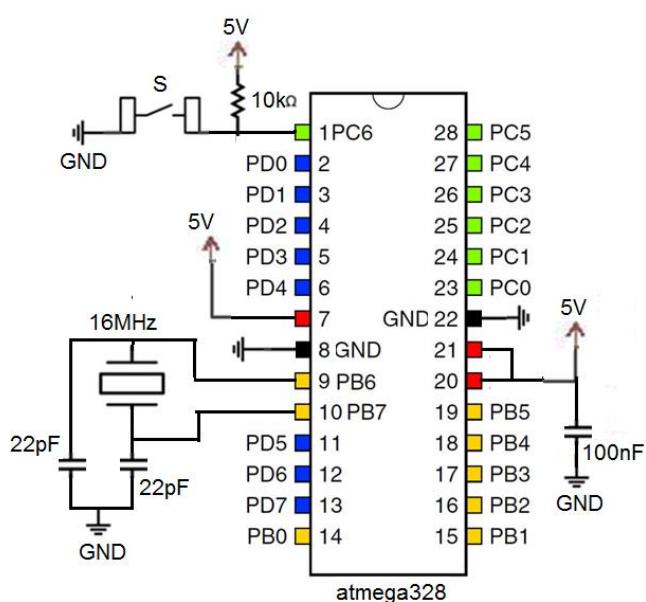


شكل ٤-٤ نظرة عامة على أطراف المتحكم  
atmega328

١٨ - كل أطراف الإدخال والإخراج يمكنها أن تتبع أو تعطى تيارا حتى ٢٠ ميلي أمبير، وهذا كافى جدا لتشغيل الكثير من التطبيقات مثل لمبات البيان LEDs دون الحاجة إلى الاستعانة لدافع تيار لتشغيل هذه الأحمال.

شكل ٤-٤ يبين أطراف هذا المتحكم، حيث نلاحظ أن الشريحة لها ٢٨ طرفا، منهم ٢٣ طرفا خاصة بإدخال وإخراج البيانات ووظائف أخرى كما سنجري ذلك عند الشرح التفصيلي للوظائف المختلفة لهذا المتحكم. الخامس خطوط الباقي منها اثنان للأرضي GND واثنان للقدرة Vcc، وجهد آخر Vref وهو الجهد المرجعى للمحول التماشى الرقمى، وسيأتى شرح ذلك بالتفصيل مع شرح المحول التماشى الرقمى.

شكل ٤-٣ يبين تجهيز المتحكم من حيث توصيل القدرة له وتوصيل مصدر النبضات فى حالة استخدام مصدر خارجي لنبضات التزامن. خط القدرة Vcc هو خط القدرة العام للمتحكم وكذلك الأرضى GND. أما خط القدرة AVcc فهو خط القدرة الخاص بالمحول التماشى الرقمى ADC، والذى يجب أن يوصل عليه مكثف تتعيم حوالى ٠.١



شكل ٣-٤ إعداد المتحكم للعمل (توصيل القدرة والتزامن)

ميكروفاراد لضمان تقليل تأثير ضوضاء مصدر القدرة على المحول ADC. يمكن توصيل كل الأطراف  $V_{CC}$  على نفس مصدر القدرة ٥.٥V مثلا، إلا إذا كان هناك ضرورة لفصل مصدر القدرة الخاص بالمحول ADC، غالبا لا يكون هناك داعي لهذا الفصل في معظم التطبيقات. الجهد  $V_{ref}$  أيضا يمكن توصيله مع الجهد  $V_{CC}$  إلا إذا كان هناك ضرورة للفصل وغالبا لا توجد هذه الضرورة أيضا. طرف الأرضي GND يمكن توصيلهما مع بعضهما أيضا. في حالة توصيل نبضات تزامن خارجية كما في شكل ٣ يتم استخدام بللورة بالتردد المطلوب ١٦ ميجاهرتز مثلا

مع مكثفين كل منهما ٢٢ بيكوفاراد كما في شكل ٣. الطرف واحد يمكن استخدامه كطرف إعادة وضع reset للشريحة وذلك عن طريق توصيل هذا الطرف على مصدر القدرة ٥ فولت من خلال مقاومة ١٠ كيلوأوم كمقاومة جذب بحيث يكون هذا الطرف على الجهد طالما أن المتحكم يعمل بحالة طبيعية. عند الضغط على المفتاح S يتم توصيل الأرضي على الطرف ويحدث إعادة وضع reset للمتحكم. لاحظ أن استخدام هذا الطرف (الطرف ١) في هذا الغرض يعني أن هذا الخط لن يمكن استخدامه كخط إدخال أو إخراج للبيانات بعد ذلك، لذلك فإنه لا يتم توصيله بهذه الطريقة إلا إذا كان يمكن الاستغناء عن هذا الخط كخط إدخال وإخراج. بذلك تصبح الشريحة جاهزة من حيث القدرة ونبضات التزامن للتوصيل على أي دوائر خارجية أخرى.

#### ٤-٣ إدخال وإخراج البيانات للمتحكم atmega328

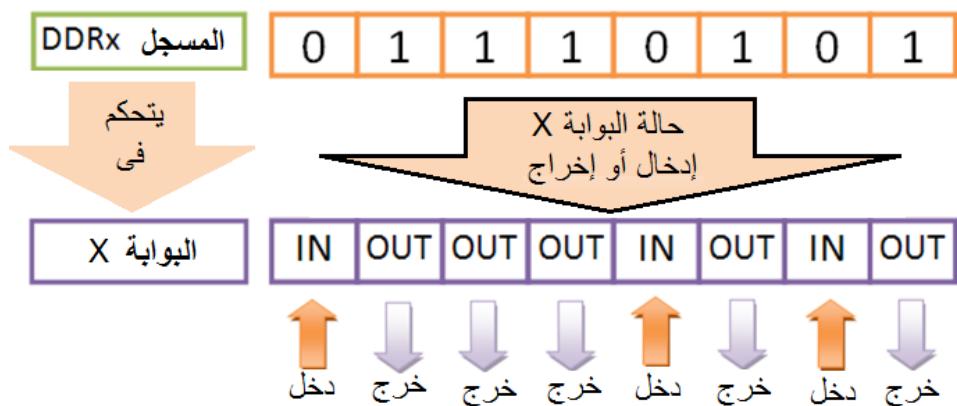
المقصود من إدخال البيانات إلى أي متحكم أو معالج هو كيفية جعل المتحكم يقرأ بيانات من أجهزة موصولة عليه مثل لوحة مفاتيح مثلا، أو قراءة حساس أو مستشعر لدرجة الحرارة بعد تحويلها إلى الصورة الرقمية، وهكذا الكثير من الإشارات التي تحتاج لإدخالها إلى المتحكم بغرض التحكم فيها. أما إخراج البيانات فالمقصود منه إخراج بيانات من المتحكم لأحد الأجهزة الموصولة عليه من أجل التحكم في هذا الجهاز، مثل إخراج إشارة ترفع أو تخفض من سرعة

موتور، أو إخراج إشارة من أجل عرضها على لوحة عرض بللورة مائية LCD، وهكذا الكثير من إشارات التحكم التي نحتاجها من المتحكم.

بالنظر إلى شكل ٤ الذي يبين نظرة عامة على أطراف الشريحة atmega328 سنجد أن بها ثلاثة بوابات يمكن من خلالهم إدخال أو إخراج البيانات الرقمية وهي كالتالي: البوابة PB وهي ٨ أطراف باللون الأصفر (٨ بت) من PC، البوابة PC وهي ٧ أطراف فقط باللون الأخضر من PC0 حتى PC6، ثم البوابة PD وهي ٨ أطراف باللون الأزرق من PD0 حتى PD7، وهذه البوابات الثلاثة تشكل أطراف الإدخال والإخراج التي ذكرنا من قبل أن عددها ٢٣ طرفا ( $8+7+8=23$ ). هذه البوابات الثلاثة قابلة للبرمجة أو للتشكيل، بمعنى أنه يمكن برمجة كل طرف من أطرافها بحيث يصبح خط إدخال للبيانات أو خط إخراج للبيانات، حتى أنه يمكنك أن تجعل كل الـ ٢٣ خط تمثل خطوط إدخال، أو كل الـ ٢٣ خط تمثل خطوط إخراج للبيانات أو أي تشكيله بين الإدخال والإخراج. يتم التعامل مع كل واحدة من هذه البوابات من خلال ثلاثة مسجلات كالتالي:

## ١ - المسجل DDRx

هذا المسجل هو مسجل اتجاه البيانات Data Direction Register لأى بوابة x من البوابات الثلاث، حيث x تعنى البوابة B أو C أو D. مثلا DDRB يعني مسجل اتجاه البيانات للبوابة B. هذا المسجل يتكون من ٨ بت، ووضع ١ في أى بت من بتات هذا المسجل يجعل طرف البوابة المقابل لهذه البت طرف خرج، ووضع ٠ في أى بت من بتات هذا المسجل يجعل طرف البوابة المقابل لهذه البت طرف دخل. شكل ٤ يبين رسميا توضيحا لذلك.



شكل ٤ المسجل DDRx يتحكم في تشكيل أطراف البوابة X ليكون كل طرف دخل أو خرج

في شكل ٤-٤ تم وضع الرقم الثنائي 01110101 في مسجل التحكم في الاتجاه DDR<sub>X</sub> ليتحكم في اتجاه كل طرف من أطراف البوابة X. البت التي بها 1 في المسجل DDR<sub>X</sub> يجعل الطرف المقابل في البوابة X طرف خرج، والبت التي بها 0 في المسجل DDR<sub>X</sub> يجعل طرف البوابة X المقابل طرف دخل. تذكر أن X هنا تعنى أي واحدة من البوابات الثلاث. باستخدام لغة C يمكن كتابة الأمر الذى سيجعل أطراف البوابة D كما هو موضح في شكل ٤-٤ كالتالى:

DDRD=0b01110101

وذلك بكتابة الرقم 01110101 بالصورة الثنائية، لذلك سبقناه بصفر ثم الحرف b حيث b تعنى binary. يمكن كتابة نفس الأمر السابق بوضع الرقم في صورته المستعشرية hexadecimal كالتالى:

DDRD=0x75

حيث 0x تعنى أن الرقم التالي له في صورته المستعشرية. سنرى ذلك بالتفصيل بعد قليل بعد قليل مع البدأ في البرمجة في وسط البرنامج أتمل استديو. لاحظ أن تشكيل أطراف أي بوابة تكون دخلاً أو خروجاً سيكون في بداية البرنامج كما سنرى بعد قليل.

## ٢- المسجل PORT<sub>x</sub>

المسجل PORT<sub>x</sub> يحتوى البيانات المطلوب إخراجها على البوابة X. هذا المسجل بالطبع يتكون من ٨ بتات. كمثال على ذلك يمكن إخراج الرقم F0 المستعشرى على أطراف البوابة D بالأمرتين التاليين:

DDRD=0xFF;

POR TD=0xF0;

حيث الأمر الأول يجعل أطراف البوابة D كلها أطراف خرج، والأمر الثاني سيضع أربع أصفار على الأطراف الأربع الأولى من البوابة D وأربع وحيد على الأطراف الأربع الأخيرة من نفس البوابة.

## ٣- المسجل PIN<sub>x</sub>

المسجل PIN<sub>x</sub> هو مسجل ٨ بت يحتوى البيانات الموجودة على أطراف البوابة X، بعد أن تكون هذه البوابة قد تم تشكيلها كبوابة دخل، أي أن هذا المسجل يحتوى باختصار البيانات المدخلة على أطراف أي بوابة. كمثال على ذلك أنظر للأوامر الأربع التالية:

DDRD=0x00;

DDRB=0xFF;

n=PIND;

PORTRB=n;

حيث الأمر الأول يشكل أطراف البوابة D لتكون كلها أطراف دخل، والأمر الثاني يشكل أطراف البوابة B لتكون كلها أطراف خرج. الأمر الثالث يقرأ البوابة D ويوضع محتوياتها في المتغير n، والأمر الرابع يخرج محتويات المتغير n على أطراف البوابة B. سيكون هناك تفاصيل أكثر عن البرمجة بعد قليل، ولكن دعنا الآن نلقي نظرة تفصيلية على كيف يتم تشكيل أي طرف من أطراف أي بوابة كطرف دخل أو خرج داخل المتحكم نفسه.

### تشكيل أي طرف من أطراف أي بوابة كطرف خرج

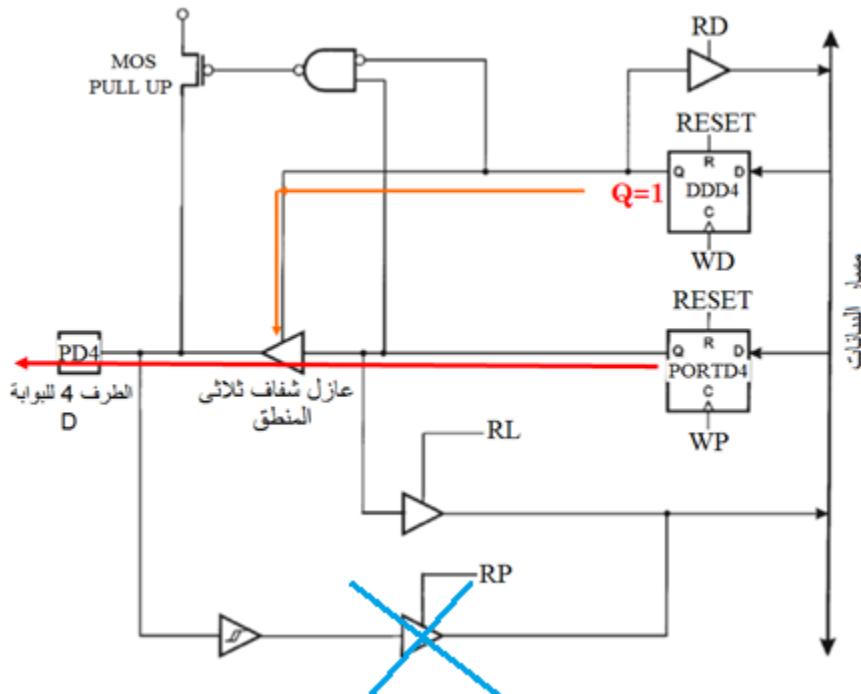
شكل ٤-٥ يبين طريقة تشكيل الطرف ٤ من أطراف البوابة D كطرف خرج كمثال على أي طرف آخر من أطراف أي بوابة، أي أن هذه المكونات ستكون مكررة ثمان مرات لكل بوابة داخل المتحكم. لكي نشكل الطرف ٤ من البوابة D كطرف خرج يجب وضع ١ في البت المقابلة لهذا الطرف في مسجل تحديد الاتجاه DDRD، أي أننا سنجعل DDD4=1. تذكر جيداً أن هذه البت هي في النهاية عبارة عن قلاب flip flop ومعنى أن نضع فيه ١، أننا سنجعل خرج هذا القلاب Q=1 كما في شكل ٤-٥. خرج القلاب متصل بخط تحكم العازل الشفاف ثلاثي المقطع بحيث أنه عندما يكون ١ فإن العازل سيكون موصلاً (دخله يساوى خرجه ومن هنا تأتي كلمة شفاف)، وعندما يكون خط التحكم يساوى ٠، فإن العازل سيكون مفتوح (الخرج لا يرى الدخل open circuit). عندما يكون العازل موصلاً (دخله يساوى خرجه) فإنه سيجعل خرج بت مسجل الخرج PORTD4 موصلاً على طرفيها الخارجي PD4. بذلك فإن أي بيانات يتم وضعها في البت PORTD4 سيتم رؤيتها مباشرة على طرف البوابة الخارجي PD4.

في الشكل ٥ بتنشيط الخط RD يمكن قراءة محتويات القلاب DDD4، وكذلك بتنشيط الخط RL يمكن قراءة خرج القلاب PORTD4 وهناك أوامر لتنفيذ ذلك. لاحظ أن العازل الشفاف الأسفلي الموجود في طريق القراءة سيكون غير موصلاً لأن الإشارة RP تساوى صفر في هذه الحالة.

### تشكيل أي طرف من أطراف أي بوابة كطرف دخل

شكل ٤-٦ يبين طريقة تشكيل الطرف ٤ من أطراف البوابة D كطرف دخل كمثال على أي طرف آخر من أطراف أي بوابة. لكي نشكل الطرف ٤ من البوابة D كطرف دخل يجب وضع ٠ في البت المقابلة لهذا الطرف في مسجل تحديد الاتجاه DDRD، أي أننا سنجعل DDRD4=0. خرج القلاب متصل بخط تحكم العازل الشفاف ثلاثي

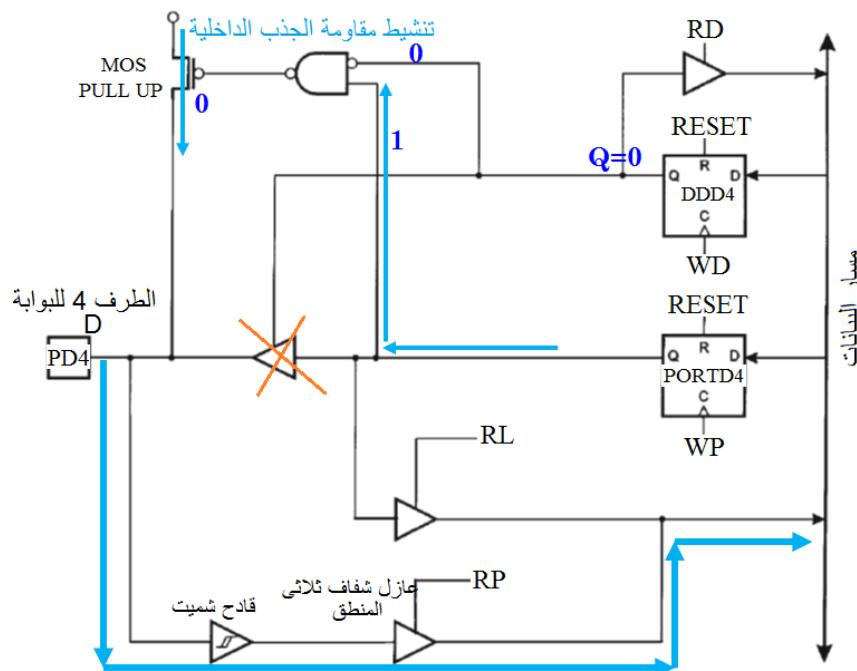
المنطق بحيث عندما يكون خط التحكم يساوى 0 ، فإن العازل سيكون مفتوحا (الخرج لا يرى الدخل open circuit)، وبذلك فإن بิต المسجل PORTD4 لن تكون موصولة على طرف الخرج PD4 كما كان في الحالة السابقة. في هذه الحالة ستكون الإشارة RP (والتي تعنى Read Port) تساوى واحد وهذا الواحد سيجعل العازل الشفاف ثلاثي المنطق (في أسفل الشكل) موصلًا (خرجه يساوى دخله) وبذلك فإن الطرف 4 يساوى دخله وينتهي إلى مسار البيانات بحيث تذهب محتوياته إلى البت PIND4 في مسجل القراءة PIND للبوابة D، وبذلك تتم قراءة الطرف 4 للبوابة .D



شكل ٤-٥ تشكيل الطرف ٤ من البوابة D كطرف خرج

عند توصيل أي إشارة لطرف دخل في أي بوابة، فإنه يجب توصيل هذا الطرف على مقاومة جذب pull up resistance. هذه المقاومة يمكن توصيلها خارجيا باستخدام مقاومة ١٠ كيلوأوم تقربيا يتم توصيلها من خارج المتحكم على مصدر القدرة VCC. المتحكمات AVR تتيح لك توصيل هذه المقاومة من داخل المتحكم نفسه عن طريق تنشيط الموسفيت الموجود في أعلى يسار الشكل ٤-٦ عن طريق وضع 0 على قاعدة هذا الموسفيت فيصبح موصلًا ويساوى مقاومة تصل بين مصدر القدرة VCC وطرف البوابة PD4. لكنك تحصل على 0 عند قاعدة الموسفيت فإن البوابة NAND الموصولة على قاعدته يجب أن يكون دخليها كل منها يساوى واحد، وهذا هو الحادث فعلاً

شكل ٦-٤ حيث أحد دخلي البوابة NAND متصل على خرج القلاب DDD4 الذي يساوى صفر وهو موصل من خلال عاكس، وأما الدخل الآخر لبوابة الناند فإنه يأتي من خرج القلاب PORTD4 والذي يجب أن نخرج عليه واحد لكى نضمن أن خرج بوابة الناند ستكون صفراء. الخلاصة من ذلك أننا لكي ننشط مقومة الجذب الداخلية مع أي طرف دخل لأى بوابة فإنه يجب إخراج واحد على هذا الطرف من البوابة، وسنرى ذلك في الكثير من الأمثلة القادمة. قادح الأشميット الموجود في طريق إشارة الدخل موجود لتحسين شكل إشارة الدخل التي يتم قراءتها من على الطرف PD4.



شكل ٤-٦ تشكييل الطرف ٤ من البوابة D كطرف دخل

## ٤-٤ برنامج المحاكاة أتمل استديو

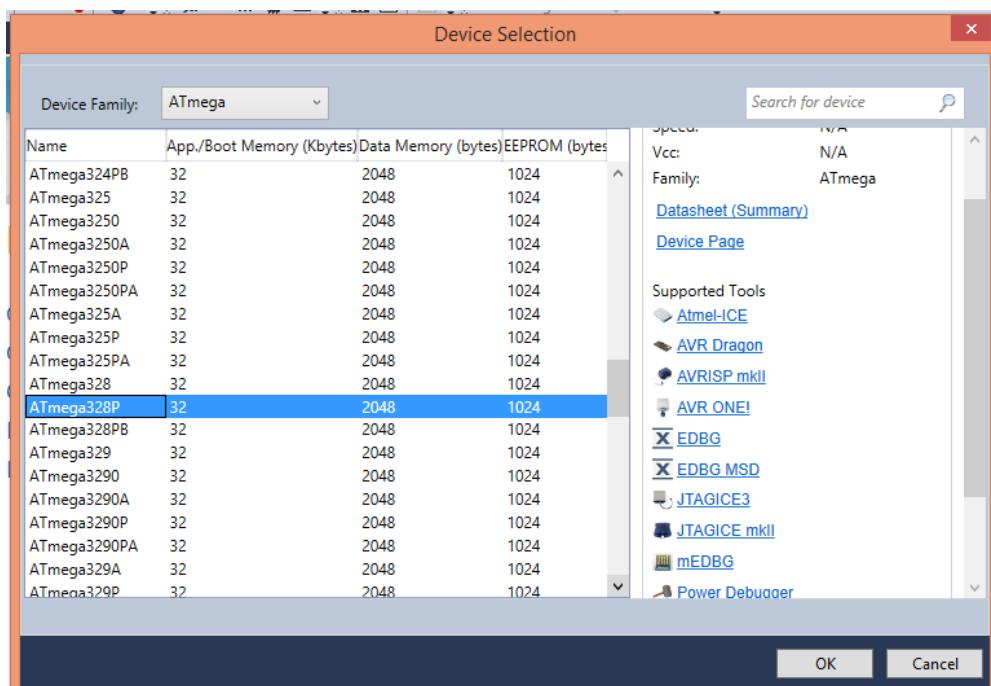
نؤكد هنا على أنه هناك الكثير من برامج التحرير والمحاكاة، وننصح بأن من تعود على أحد البرامج مسبقاً فعليه أن يستمر مع هذا البرنامج وبالذات إذا كان مستريحاً في التعامل معه، أما إذا لم تكن تعاملت من قبل مع أي واحد من هذه البرنامج فننصح بالبدأ مع برنامج أتمل استديو لما به من الكثير الإمكانيات وبالذات في التعامل مع المشاريع الكبيرة. هناك أكثر من إصدار من إصدارات الأتمل ستديو آخرها هو الإصدار ٧ الذي سنعمل عليه في إعداد هذا الكتاب. هذا الإصدار وأي إصدار سابق يمكنك تنزيله مجاناً من على موقع شركة إنتل وهو الموقع التالي: [http://www.atmel.com/microsite/atmel\\_studio7/](http://www.atmel.com/microsite/atmel_studio7/). بعد تنزيل البرنامج على الحاسوب الخاص بك، إتبع الخطوات التالية حتى تبدأ تحرير البرنامج البسيط الذي سيضيء لمبة بيان بناء على فتح أو قفل مفتاح كما سترى، كبرنامج بسيط نفهم منه فقط إمكانية التعامل مع الأتمل استديو والمحاكاة لهذا البرنامج لنرى صحة تشغيله قبل أن نأخذه على برنامج المحاكاة الإلكتروني لنضع البرنامج الستعشري من أجل التشغيل الفعلى على شريحة المتحكم .atmega328

١- بالنقر على أيقونة الأتمل استديو ٧ ستظهر لك شاشة البدأ المبينة في شكل ٤-٧.



شكل ٤-٧ شاشة بدأ الأتمل ستديو ٧

٢- أنقر على وصلة فتح مشروع جديد ... New Project المبينة في شكل ٤-٧، حيث ستظهر لك عدة اختيار أنقر على GCC C Executable Project وهو الثاني في القائمة، وفي أسفل الشاشة ستجد مساحة لكتابة إسم البرنامج أو المشروع الذي سيتم فتحه، في حالتنا هذه كتبنا LED1 ليكون إسم أول مشروع. في المساحة التالية سيقترح عليك الأتمل ستديو إسم المجلد الذي سيتم تخزين المشروع فيه وهو سيكون المسار التالي: c:\users\administrator\Documents\Atmel Studio\7.0، وهو مجلد على الاسطوانة C في ال Document، ونعتقد أن هذا أفضل اختيار، لذلك اتركه كما هو ثم انقر على OK. بعد ذلك ستظهر لك شاشة لاختيار المتحكم الذي ستتعامل معه من بين العديد من المتحكمات كما في شكل ٤-٨. لتسهيل الوصول للمتحكم المراد، اختار العائلة atmega، أولاً حيث ستظهر لك كل المتحكمات الموجودة في هذه العائلة. إختار من هذه العائلة المتحكم atmega328p حيث بمجرد النقر عليه يتم تعليمه ويظهر على يمين النافذة بيانات إضافية عن هذا المتحكم ومنها مثلاً كتالوج ملخص له وجميع الأدوات التي يمكن أن يتعامل معها هذا المتحكم ومن ضمن هذه الأدوات المحاكى simulator. انقر OK لتنقل إلى شاشة المشروع الجديد.



شكل ٤-٨ اختيار المتحكم الذي سيتم التعامل معه

٣- شاشة المشروع الجديد عبارة عن مجموعة من النوافذ، أهمها نافذة البرمجة وقد ظهر فيها نموذج مقترح للبرنامج يمكنك البدأ في تطويره أو البناء عليه، هذا البرنامج النموذج سعيد كتابته هنا كما يلى:

```
/*
 * LED1.c      إسم البرنامج الذى تم اختياره
 *
 * تاريخ إنشاء هذا المشروع   * Created: 6/4/2017 5:39:49 PM
 * إسم مؤلف البرنامج (ضع إسمك)   * Author : M. Eladawy
 */
#include <avr/io.h>
int main(void)
{
    /* Replace with your application code */
    while (1)
    {
    }
}
```

ربما يكون محترف البرمجة في وسط البرمجة الخاص بالفي gioval استديو يو متعدون على هذا الشكل. كل الكتابة الخضراء هي مجرد تعليقات يمكنك أن تكتب فيها ما شئت وبأى لغة ومنها إسم البرنامج، وتاريخ إصداره، وإسم مؤلف هذا البرنامج. هذه الأسطر تكون في بداية نموذج كل برنامج جديد، وتكون مصورة بين السطرين `/*...*/`. يأتي بعد ذلك التعبير `#include <avr/io.h>` وهو لتضمين الملف `avr/io.h` مع برنامحك لأنه يحتوى على كل بيانات شرائح المتحكمات AVR ومن أهمها أسماء المسجلات التي سيتم استخدامها، وهذا الأمر ضروري وجوده في كل برامح المتحكمات AVR. يمكن تضمين ملفات أخرى في هذا الموضع عند التعامل مع مكونات محددة داخل المتحكم مثل المقاطعات والمؤقتات كما سنرى فيما بعد. يأتي بعد ذلك التعبير `int main(void)` وهو بداية أي برماج يتم كتابته في وسط البرمجة بلغة C حيث `main` تعنى بداية البرنامج الأساسي، و `int` تعنى أن هذا البرنامج يعطى أو يعود بقيمة صحيحة، و `void` تعنى أنه لا يوجد متغيرات يتم تمريرها إلى البرنامج `main`، ويمكن مراجعة ذلك في أي مرجع بسيط على لغة C أو أي درس من الدروس الكثيرة على الإنترت. البرنامج الأساسي `main` في لغة C لابد أن يكون بين قوسين من النوع

{ .... } وهذا هو سبب وجود القوس الأول { } بعد الأمر main مباشرة والقوس الثاني { } في نهاية البرنامج.  
بعد ذلك ستتجدد العبارة التعليقية باللون الأخضر /\* Replace with your application code \*/  
والتي تعنى أن تبدء من هنا كتابة أوامر البرنامج الخاص بك، أو تستبدلها بهذه العبارة. الشيء المهم هنا والذى يجب التأكيد عليه هو وجود الحلقة اللانهائية (1) while والتي يجب أن تكون موجودة في أى برنامج. لاحظ أن هذه الحلقة يوجد القوس { } في بدايتها، كما أنها تنتهى أيضا بالقوس { } كما في البرنامج، فما السبب في وجود هذه الحلقة اللانهائية؟

### الحلقة اللانهائية في برامج المتحكمات

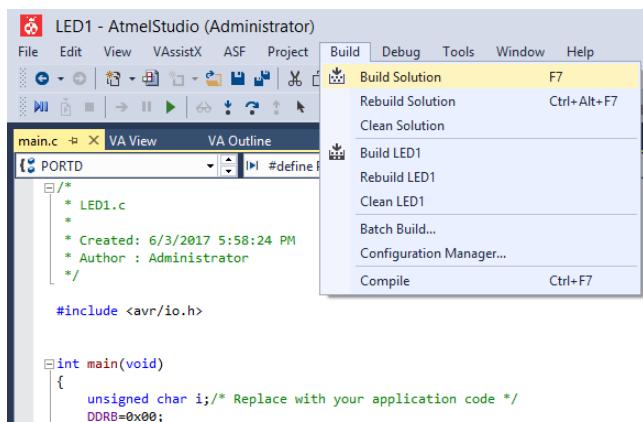
يختلف مفهوم كتابة برنامج لمتحكم يقوم بالتحكم في عملية صناعية معينة، ولتكن التحكم في سرعة موتور مثلا، عن برنامج بلغة C يحسب مرتبات موظفين مثلا في شركة من الشركات. في الوضع الأول يقوم المتحكم بقراءة سرعة المотор، ومقارنها بقيمة مرجعية لهذه السرعة، فإذا كانت السرعة الحقيقية للمotor أعلى من السرعة المرجعية فإنه يقوم بعمل إجراء ينخفض من السرعة الحقيقية، وإذا كانت السرعة الحقيقية أقل من السرعة المرجعية، فإنه يقوم بعمل إجراء يرفع من السرعة الحقيقية، وهكذا يظل برنامج المتحكم يدور في حلقة لا نهائية يقرأ السرعة الحقيقية، ومقارنها بالسرعة المرجعية، ويأخذ قرار، ولن يخرج المتحكم من هذه الحلقة إلا إذا تم إيقاف النظام بالكامل، لذلك لزم وجود حلقة لا نهائية في البرنامج في مثل هذه التطبيقات. أما برنامج حساب مرتبات الموظفين الذي يجب أن يقوم به أحد المعالجات وليس أحد المتحكمات، فإن المعالج سيحسب المرتبات مرة في أول كل شهر ثم يخرج من البرنامج لعمل أشياء أخرى أو يتوقف نهائيا عن العمل، ومعنى أن البرنامج يتوقف عن العمل أنه يتم الخروج من وسط البرمجة في لغة C والخروج إلى وسط نظام التشغيل الخاص بالحاسوب. حتى في أبسط التطبيقات، ولو أن المتحكم سيضىء LED مثلا ويتوقف، فإنه بعد أن يضئ هذا الـ LED عليه أن يدخل في حلقة لا نهائية لا يعمل فيها شيء مثل الحلقة { } (1) while في البرنامج النموذج السابق، حيث ستكون هذه هي طريقة تعطيل المتحكم من عمل أي شيء يؤثر في التطبيق، وفي نفس الوقت يحافظ على عمل التطبيق، حيث لا مجال هنا للخروج من الحلقة اللانهائية إلى نظام التشغيل.

٤ - في شاشة البرمجة سنطور البرنامج النموذج إلى أول برنامج سنقوم بتنفيذه في هذا الكتاب. إسم هذا البرنامج سيكون LED1 وستكون شاشة تحريره كالتالي:

```

/*
* LED1.c
*
* Created: 6/3/2017 5:58:24 PM
* Author : M. Eladawy
*/
#include <avr/io.h>
int main(void)
{
    unsigned char i;
    DDRB=0x00;
    DDRD=0xFF;
    while (1)
    {
        i=PINB;
        PORTD=i;
    }
}

```



شكل ٤-٩ بناء البرنامج للتحقق من صحته لغويًا

من الممكن كتابته في النظام الثنائي مباشرة كالتالي DDRB=0b00000000. الأمر التالي هو تخصيص البوابة D لتكون بوابة إخراج عن طريق وضع وحيد في مسجل الاتجاه الخاص بها باستخدام الأمر التالي:

بدأنا البرنامج الأساسي بتعريف متغير `i` ليكون متغير حرف `char` مكون من 8 بت، وبدون إشارة `unsigned`. بعد ذلك تم تخصيص البوابة B (جميع أطرافها) لتكون بوابة إدخال عن طريق وضع أصفار في مسجل الاتجاه الخاص بها باستخدام الأمر `DDRB=0x00`, حيث `0x00` تعني أن هذا الرقم مثل في النظام المستعشرى، أي أن هذا الرقم سيكون 00000000، وكان

بعد ذلك دخل البرنامج كما ترى في حلقة لا نهاية باستخدام الأمر (1) while. في داخل هذه الحلقة يقوم البرنامج بقراءة محتويات البوابة B ووضعها في المتغير i، ثم

يتم إخراج محتويات المتغير i على أطراف

البوابة D من خلال الأمر

PORTD=i، وبذلك فإن البرنامج

باختصار يقرأ محتويات البوابة B ويضعها

على البوابة D وذلك إلى مالا نهاية. نريد

الآن تنفيذ البرنامج لترى هل يتم تنفيذه

بطريقة صحيحة أم لا.

٥- قبل البدأ في تنفيذ البرنامج ومحاكاته يجب

أن نخزنه عن طريق النقر على قائمة الملفات

ومنها نقر على الاختيار save all لتخزين كل ما قمنا بتحريره أو أضفناه للبرنامج.

٦- للتأكد من صحة البرنامج وأنه لا يحتوى أى أخطاء لغوية (أخطاء لغة C) فإننا نقوم ببناؤه عن طريق النقر

على قائمة Build ومنها اختيار Build Solution أو نقر على F7 بدلاً من ذلك، كما هو موضح في

شكل ٩-٤.

٧- بتنفيذ الخطوة ٦ السابقة سيدأ الاستديو في بناء النظام وسترى ذلك في ظهور شاشة جديدة تسمى شاشة

الخرج output يظهر فيها حالة البناء وفي نهاية هذه الشاشة يجب أن ترى العبارة الدالة على نجاح البرنامج

وعدم احتواه أى أخطاء كما في شكل ٤-١٠. إذا ظهرت أى أخطاء ستظهر لك الأوامر التي تحتوى هذه

الأخطاء، وفي هذه الحالة عليك العودة إلى شاشة تحرير البرنامج وتصحيح هذه الأخطاء، ثم إعادة البناء،

وهكذا إلى أن تحصل على العبارة الدالة على نجاح البرنامج وخلوة من الأخطاء كالموضحة في شكل ٤-١٠.

٨- بعد التأكد من خلو البرنامج من الأخطاء اللغوية كما في شكل ٤-١٠، نريد الآن التأكد من أنه سيعمل

بطريقة صحيحة فعلاً وأنه سيقرأ البوابة B ويضع محتوياتها على البوابة D. يتم ذلك بالنقر على قائمة Debug

ومنها نقر على الخيار Start Debug and Break أو النقر على Alt+F5 كما في شكل ٤-١١. القائمة

Debug الموضحة في شكل ٤-١١ تحتوى الكثير من الاختيارات والتي من أهمها الاستمرار Continue أو

F5 التي تبدأ التنفيذ من نقطة توقف معينة Break. هناك أيضاً الاختيار Step Into أو F11 والتي بالنقر

عليها يتم تنفيذ الخطوة التي توقف عنها عملية التنفيذ حتى لو أدى ذلك إلى الدخول في برنامج جانبي. هناك

```

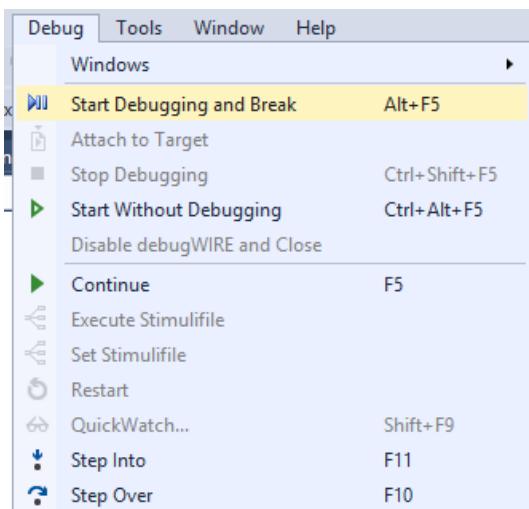
Output
Show output from: Build
Done executing task "RunCompilerTask".
Using "RunOutputFileVerifyTask" task from assembly "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Atmel\Tasks\RunOutputFileVerifyTask"
    Program Memory Usage      : 144 bytes  0.4 % Full
    Data Memory Usage        : 0 bytes   0.0 % Full
Done executing task "RunOutputFileVerifyTask".
Done building target "CoreBuild" in project "LED1.cproj".
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '')
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Atmel\Tasks\Build.cs" succeeded.
Done building target "Build" in project "LED1.cproj".
Done building project "LED1.cproj".

Build succeeded.
=====
Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==
1

```

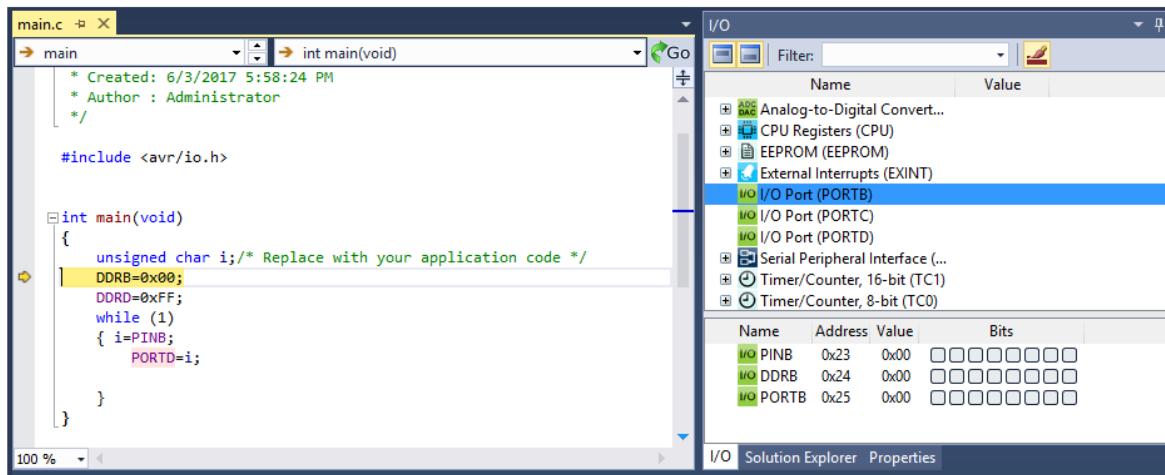
شكل ٤-١٠ نجاح بناء البرنامج

أيضا الاختيار Step Over F10 الذى ينفذ خطوة واحدة مع اعتبار أى برنامج جانبة خطوة واحدة. هناك الكثير من الخيارات الأخرى التى يمكنك اللعب معها كما سرى في الخطوات التالية.



شكل ١١-٤ قائمة تبع تنفيذ البرنامج Debug

٩- هناك الكثير من النوافذ الجانبية المهمة التي سنحتاجها في عملية محاكاة البرنامج ومن أهمها نافذة الإدخال والإخراج التي تعرض بوابات الإدخال والإخراج في المتحكم الذى تتعامل معه والكثير من الملحقات الأخرى. كل هذه النوافذ يمكنك إظهارها بالأحجام التي تريدها وفي الموضع الذى تريدها. الشريط الأسفل تحت شريط قوائم الاستديو يحتوى الكثير من الأيقونات التي بالنقر عليها تصل إلى ما تزيد أسرع ومنها النوافذ الجانبية التي تزيد إظهارها، وبالوقوف بالماوس على أى واحدة من هذه الأيقونات يظهر لك وظيفة هذه الأيقونة.



شكل ١٢-٤ نافذتي البرمجة والإدخال والإخراج للبرنامج LED1.

١٠- بتنفيذ الخطوة ٨ وإظهار نافذة الإدخال والإخراج كما في الخطوة ٩ سرى أن البرنامج تم تنفيذه والوقف على أول خطوة في البرنامج الأساسي main وظهور سهم أصفر على يسار قائمة البرنامج مع إظهار الخطوة الأولى وهي الخطوة التي سيتم تنفيذها بمجرد أن نضرب F10 أو ننفذ خطوة واحدة. شكل ١٢-٤ يبين

نافذة البرمجة ونافذة الإدخال والإخراج وقد تم النقر على البوابة B لكي نرى تأثير تنفيذ أول أمر. لاحظ أنه بالنقر على اختيار البوابة B في نافذة الإدخال والإخراج (I/O Port) يظهر لك في أسفل النافذة المسجلات الثلاثة الخاصة بهذه البوابة وهي كما ترى في الشكل وكما ذكرنا من قبل PINB وهو مسجل

Name	Address	Value	Bits
PINB	0x29	0x00	oooooooooooo
DDRD	0x2A	0xFF	oooooooooooo
PORTB	0x2B	0x00	oooooooooooo

شكل ١٣-٤ وضع وحيد في المسجل DDRD

Name	Address	Value	Bits
PINB	0x23	0xF	oooooooooooo
DDRD	0x24	0x00	oooooooooooo
PORTB	0x25	0x00	oooooooooooo

شكل ١٤-٤ التسجيل في المسجل PINB

لكي نرى ذلك سنذهب إلى نافذة الإدخال والإخراج وننقر على البوابة D، حيث سنرى أن محتويات جميع مسجلات البوابة D أيضاً تساوى أصفاراً لأننا لم ننفذ الأمر الثاني حتى الآن.

الإدخال الذى يحتوى البيانات الدخلة على أطراف هذه البوابة، والمسجل DDRB وهو مسجل التحكم في أطراف البوابة B، وأخيراً المسجل PORTB الذى يحتوى البيانات المراد إخراجها على أطراف هذه البوابة. ستري في الشكل أن محتويات كل هذه المسجلات الثلاثة أصفاراً لأننا مازلنا في بداية تشغيل المتحكم وهذه هي المحتويات التلقائية في بداية التشغيل .reset

١١- بضرب المفتاح F10 لتنفيذ الخطوة الأولى من البرنامج لن نرى أي تغيير على حالة البوابة B لأن الأمر الأول كان يضع أصفاراً في المسجل DDRB ولذلك لن نرى تغيير.

سنرى أيضاً أن التنفيذ (السهم الأصفر) في نافذة البرمجة قد انتقل إلى الأمر الثاني وهو DDRD=0xFF، وهذا الأمر سيجعل محتويات مسجل الاتجاه للبوابة D يساوى وحيد.

١٢- لكي نرى ذلك سنذهب إلى نافذة الإدخال والإخراج وننقر على البوابة D، حيث سنرى أن محتويات جميع مسجلات البوابة D أيضاً تساوى أصفاراً لأننا لم ننفذ الأمر الثاني حتى الآن.

١٣- إضرب المفتاح F10 لترى نتيجة تنفيذ هذه الخطوة على مسجلات البوابة D. ستري في الحال انتقال التنفيذ إلى الأمر التالي DDRD=i، وأن المسجل PINB في نافذة الإدخال والإخراج أصبحت حمراء كما في شكل ١٣-٤ مما يعني تسجيل وحيد في هذا المسجل.

٤ - الآن الأمر الذى عليه الدور في التنفيذ هو الأمر  $PINB=i$ ، ولكن نرى نتيجة تنفيذ هذا الأمر سنذهب مرة ثانية للبوابة B في نافذة الإدخال والإخراج بالنقر عليها لإظهار مسجلاً لها. ستري أن محتويات كل مسجلاتها تساوى أصفاراً حتى الآن. الآن نريد أن نضع بيانات معينة على دخل البوابة B وسيكون ذلك بالنقر على بิต المسجل PINB في نافذة الإدخال والإخراج لنجعل محتويات هذا المسجل تساوى ٠٠٠٠١١١١١١٠٠٠ بالنقر على أول ٤ بت لجعلها وحيدة كما في شكل ٤-٤، حيث ستجد أن هذه البتات أخذت اللون الأزرق الغامق.

٥ - إضرب F10 ليتم تنفيذ الأمر  $PINB=i$  وقراءة محتويات البوابة B والانتقال إلى الأمر التالي وهو  $PORTD=i$ .

٦ - قبل تنفيذ الأمر  $PORTD=i$  أنقر على البوابة D مرة ثانية لنرى نتيجة تنفيذ هذا الأمر. لاحظ أنه قبل تنفيذ هذا الأمر فإن محتويات المسجل PORTD تساوى أصفارى، وأن محتويات المسجل DDRD أصبحت باللون الأزرق الغامق مما يدل على عدم تغيير محتوياتها.

٧ - إضرب F10 لترى نتيجة تنفيذ الأمر  $i=PORTD$ . إنك ستري في الحال أن محتويات المسجلين

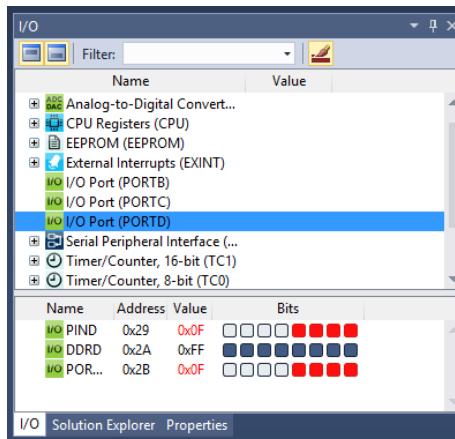
$PIND=00001111$  و  $PORTD=00001111$

أيضاً كما في شكل ٤-٥. إن ذلك يعني أن البرنامج قد قرأ المسجل PINB ووضع محتوياته على أطراف خرج البوابة D، وحيث أن هذه الأطراف موصولة أصلاً على المسجل PIND فقد تغيرت محتوياته هو الآخر.

٨ - يمكنك الآن اللعب بوضع أي بيانات تختارها على المسجل PINB وتلاحظ انتقالها إلى المسجل PORTD، مما يعني قراءة البوابة B ووضع محتوياتها على البوابة D.

٩ - أنقر على البوابة B وغير محتويات المسجل PINB لتصبح  $PINB=11110000$  كما في شكل ٦-٤ ونفذ

الأمرتين اللذين في الحلقة المغلقة لترى نتيجة تنفيذهما كما في شكل ٤-٧. من المهم جداً أن نعرف طريقة الأقل استديو في تلوين محتويات المسجلات المختلفة لكي نعرف ماذا حدث في شكل ٤-٧. المحتويات التي لا تتغير بسبب تنفيذ أي أمر من أوامر البرنامج تأخذ اللون الأزرق الغامق بدليل أن محتويات المسجل DDRD على وضعها لم تتغير، ولذلك فإن كل بتاته أخذت هذا اللون الغامق. البت التي تتغير من صفر



شكل ٤-٥ الإخراج على البوابة D

إلى واحد تأخذ اللون الأحمر كما في البات الأربعة الأخيرة في المسجلين PORTD و PIND فقد تغيرت من أصفارا إلى وحيد بسبب تغيير محتويات المسجل PINB. البت الذي تتغير من واحد إلى صفر تكون محاطة فقط باللون الأحمر ومن الداخل تأخذ اللون الفاتح كما في البات الأربعة الأولى من المسجلين PIND و PORTD أيضا بسبب تغيير محتويات المسجل PINB إلى 11110000.

٢٠ - يمكنك التأكد من هذه الطريقة في تلوين المحتويات بتنفيذ نفس الأوامر مرات أخرى حيث سترى أم محتويات المسجلين PORTD و PIND أصبحت باللون الغامق مما يعني عدم تغيير محتوياتها.

٢١ - بذلك تكون قد انتهينا من محاكاة تنفيذ أول برنامج LED1، وتأكدنا من خلوه من أي أخطاء لغوية وأنه يعمل بطريقة صحيحة، فما هي الخطوة التالية في تطوير وإعداد هذا البرنامج؟

The screenshot shows the I/O window with the following data:

Name	Value
PIND	0xF0
DDRD	0xFF
POR...	0xF0

شكل ١٧-٤ مسجلات البوابة

D بعد تنفيذ الخطوة ١٩

The screenshot shows the I/O window with the following data:

Name	Address	Value	Bits
PINB	0x23	0xF0	<input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
DDRB	0x24	0x00	<input type="checkbox"/>
PORTB	0x25	0x00	<input type="checkbox"/>

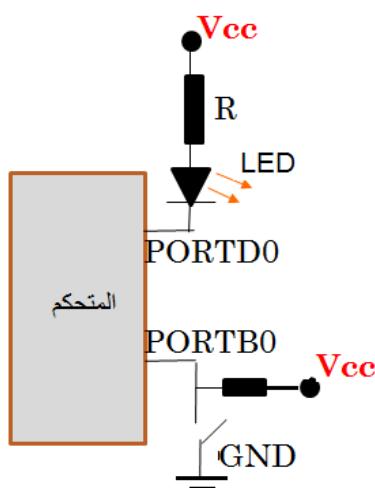
شكل ١٦-٤ المسجل

PINB=11110000

المفروض بعد أن تم محاكاة البرنامج بالطريقة السابقة والتأكد من خلوه من الأخطاء وصحة تنفيذه أن يتم وضعه (حرقه) على شريحة المتحكم من أجل اختبار النظام الكلى الذي يتحكم فيه المتحكم، ومن هنا يطرأ سؤال وهو: كيف نتأكد من صحة الدائرة الفعلية التي يعتبر المتحكم جزءا منها؟ إن برنامج الأتميل استديو أعطانا الفرصة للتأكد من صحة جزء البرمجة software من المشروع، فكيف نتأكد من الجزء الثاني من المشروع وهو الدائرة hardware. إن التأكد من صحة الدائرة الإلكترونية بما فيها المتحكم يعتبر غاية في الأهمية ولا يقل أهمية عن التأكد من صحة البرنامج، وبالذات إذا كان المشروع أو النظام المدمج embedded الذي نقوم ببناؤه معقد ويحتوى الكثير من المكونات الإلكترونية. لذلك كان لابد من وجود برنامج آخر لمحاكاة الدوائر الإلكترونية، نقوم فيه ببناء الدائرة الإلكترونية بالكامل بما في ذلك

المتحكم، ونقوم بتحميل البرنامج الذى تأكىدنا من صحته فيما سبق فى المتحكم، ثم ننفذ المشروع لنرى هل يعمل المشروع بالكامل كما هو مطلوب أم به أخطاء فى المكونات؟

## ٤-٥ برنامج محاكاة الدوائر الإلكترونية (بروتس)



شكل ١٨-٤ الدائرة التي سنختبرها

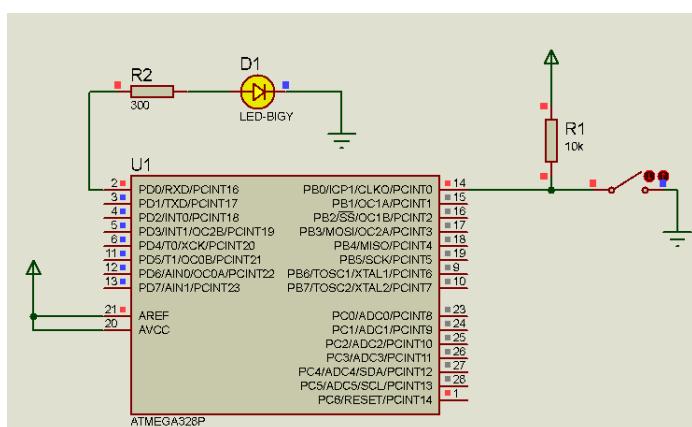
برنامجه بروتس

هناك الكثير من برامج محاكاة الدوائر الإلكترونية ومن أشهرها برنامج بروتس Protues الذى يمتاز ببساطته وسهولة تعلمه وهناك الكثير من الفيديوهات والدورس المختلفة لطريقة التعامل معه على الإنترنط، ولذلك فإننا لن نخوض في تفاصيل تشغيله في هذا المكان وسنترك للقاريء هذه المهمة التي نوصي بها لأهميتها.

في البرنامج أو المشروع البسيط الأول LED1، كانت مهمة البرنامج هي قراءة البوابة B وإخراج محتوياتها على البوابة D، ولكن تأكيد من صحة البرنامج كنا نضع بيانات بالماوس على بิต المسجل PINB ونلاحظها على المسجل PORTD من خلال نافذة الإدخال والإخراج في برنامج أتمل استديو.

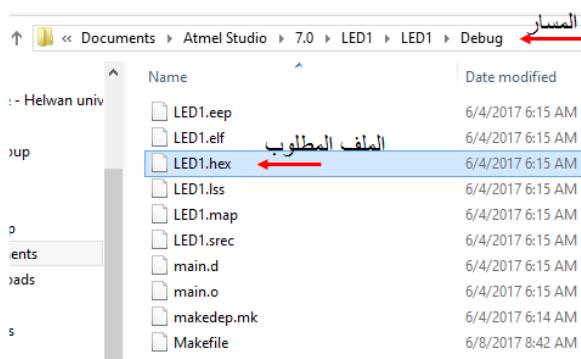
سنقوم هنا بوضع المتحكم في دائرة بسيطة مكونة من مفتاح و مصدر قدرة ومقاومة على أحد أطراف الدخل للبوابة B، بحيث أنه بالمفتوح سنضع واحد أو صفر على هذا الدخل للمتحكم كما في شكل ١٨-٤. سنضع أيضا مقاومة ودايود ضوئي Led على الطرف المقابل للبوابة D كما في الشكل.

لكى نختبر مثل هذه الدائرة البسيطة سنفتح برنامج البروتس لمحاكاة الدوائر الإلكترونية، وسنقوم بناء الدائرة كما هو موضح في شكل ١٩-٤. كما ترى في هذا الشكل فإنه عندما يكون المفتاح مفتوحا، فإن الطرف PB0



شكل ١٩-٤ تمثيل الدائرة الموضحة في شكل ١٨-٤ في برنامجه البروتس

للمتحكم يرى الجهد VCC (واحد)، وبالتالي فإن البرنامج سيقرأ هذا الواحد وينجزه على الطرف PD0 الذي سيضيء الدايوود كما هو موضح في الشكل. بالنقر على المفتاح بالماوس يصبح الطرف PB0=0، وهذا الصفر سينتقل إلى



شكل ٤ -٢٠ مسار الملف الثنائي المطلوب نقله لذاكرة البرمجة بالمتتحكم

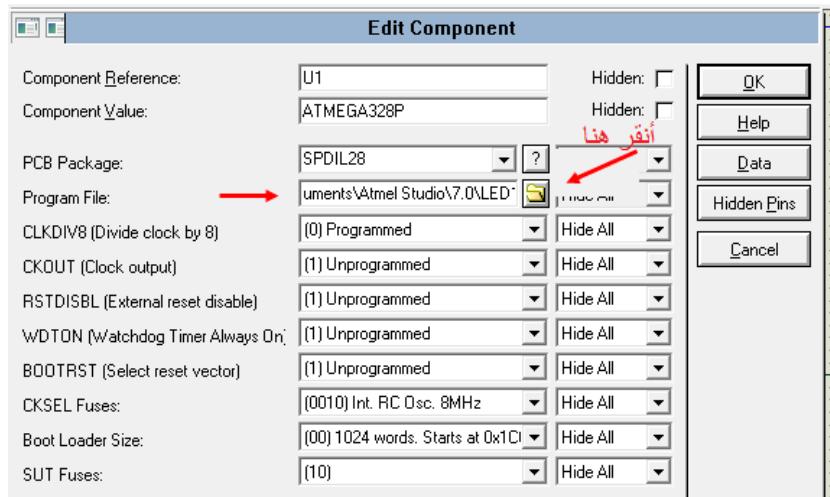
Document. بالنقر على المجلد Atmel Studio ستري مجلد آخر إسمه 7.0، وبالنقر على المجلد 7.0، ستري بداخله قائمة بالبرامج التي صممتها باستخدام أتمل استديو ٧ بحيث ستري مجلد لكل برنامج، ومن ضمنها البرنامج Led1 الذي كتبناه مسبقاً. أنقر على المجلد Led1، سيظهر لك مجلد آخر بالاسم Debug ومجدد أو مجلدات أخرى. أنقر أيضاً على المجلد Debug، سيفتح لك مجموعة مجلدات أخرى من ضمنها مجلد إسمه Debug، أنقر هنا على المجلد Debug حيث هو المجلد الذي يحتوي البرنامج Led1 في صورته الثنائية المناسبة للنقل أو الحرق في ذاكرة البرمجة للمتحكم. شكل ٤ -٢٠ يبين الملف Led1.hex معلم بالشريط الأزرق وستري المسار الذي اتبعناه في أعلى الشكل وهو Document/Atmel Studio/7.0/LED1/LED1/Debug.

المسار من حاسب آخر أو على حسب إصدار الأتمل استديو. الآن كيف ننقل هذا الملف إلى ذاكرة المتتحكم؟ سنذهب الآن إلى برنامج البروتوس وفي وجود المشروع Led1 الذي قمنا بتصميمه، أي أنها وصلنا على المتتحكم المقاومتين والمفتاح والدايوود الضوئي ووصلات الأرضي و VCC، سنقوم بالنقر مرتين على المتتحكم حيث ستظهر لك قائمة تحرير Program، حيث ستجد بجوار هذه الخاصية أيقونة فتح مجلد. أنقر هنا الأيقونة سيفتح لك البروتوس نافذة يمكنك منها أن تتبع المسار الذي تريده من خلال النقر المتتالي على مفردات المسار إلى أن تصل إلى الملف المستعشرى Led1.hex.

للمتتحكم يرى الجهد VCC (واحد)، وبالتالي فإن البرنامج ليصبح صفراء هو الآخر، وبالتالي ينطفئ الدايوود. في الشكل النقاط الحمراء التي على أطراف المتتحكم تعنى جهد عالي (١)، والنقط الزرقاء تعنى جهداً منخفض (صفر).

أهم خطوة في هذه المحاكاة هي كيفية وضع البرنامج الناتج من برنامج أتمل استديو في المتتحكم الموجود في برنامج المحاكاة بروتس حتى يمكنه تنفيذه. عندما تقوم بتنشيط برنامج أتمل استديو على حاسبك، فإنه يقوم بفتح مجلد إسمه Atmel Studio داخل مجلد الوثائق

الموضح في شكل ٤-٢٠، أنقر عليه مرتين، سيتم نقله مباشرة إلى ذاكرة المتحكم. شكل ٤-٢١ يبين نافذة خواص المتحكم التي من خلالها تتبع مسار البرنامج Led1.hex.



شكل ٤-٢١ نقل الملف المستعشرى للبرنامج إلى ذاكرة المتحكم

بهذه العملية تصبح النسخة المستعشرية للبرنامج Led1.hex وهي في الأصل النسخة الثانية مستقرة في ذاكرة البرمجة للمتحكم، ويمكنك الآن تنفيذ مشروعك في البروتوس بالنقر على أيقونة التنفيذ حيث يمكنك التعامل بالماوس مع الدائرة لتغير من حالة المفتاح (فتح وغلق) لترى تأثير ذلك على الدايوود الضوئي في الخرج.

بذلك تكون قد تأكدنا من صحة وسلامة أهم جزأين في أي مشروع وهو جزء البرمجة software الذي تأكدنا من صحته من خلال برنامج الأتمل استديو، وجزء الدائر hardware الذي تأكدنا من صحته من خلال برنامج البروتوس. الآن يمكنك شراء المكونات المختلفة بالقيم التي استخدمتها في البروتوس (قيم المقاومات وأنواع الدايوود الضوئي وغيرها من الكثير من المكونات في حالة المشاريع الكبيرة). يمكنك الآن بناء مشروعك إما على لوحة اختبار test board، أو لوحة مطبوعة printed board، أو أي وسيلة تختارها لتنفيذ مشروعك. بذلك تبقى آخر خطوة لإتمام مشروعك وهي نقل البرنامج المستعشرى Led1.hex في ذاكرة البرمجة لشريحة المتحكم الفعلية وليس الشريحة التي استخدمناها في برنامج البروتوس كما سبق، وهذا ما سنراه في الجزء التالي.

## ٤-٦ وضع البرنامج على الشريحة الفعلية للمتحكم

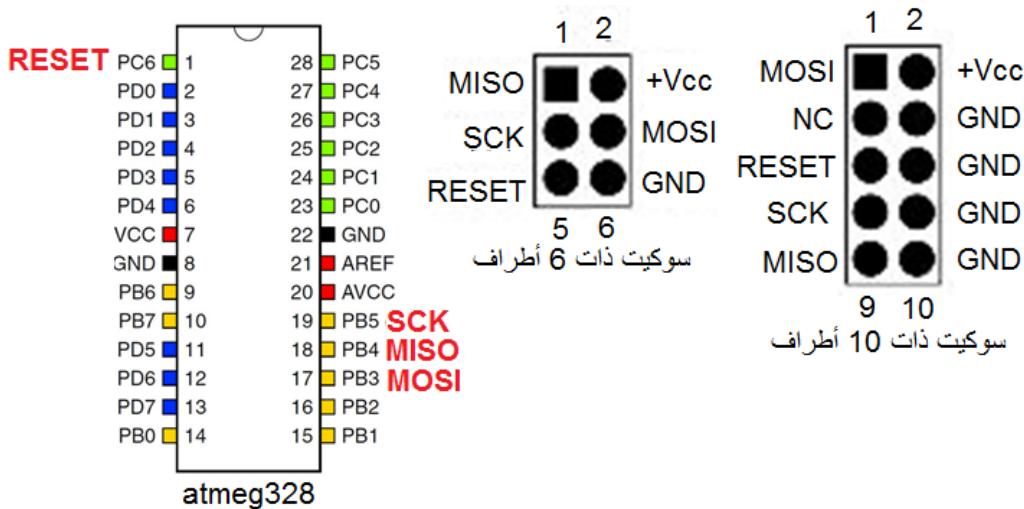
المستهدف الآن هو نقل الملف المستعشرى Led1.hex الذى نعرف مكانه في مجلد الوثائق document إلى الشريحة الفعلية للمتحكم. لكي يتم ذلك لابد من وجود وسيلة تقوم بعملية النقل للملف من الحاسب إلى الشريحة. هناك الكثير من هذه الوسائل التي تبدأ من دائرة بسيطة تقوم أنت ببناؤها واستعمالاها، والإنترن特 مليئة بمثل هذه الدوائر، إلى دوائر جاهزة تشتريها من السوق مع برنامج بسيط يكون ملحق بهذه الدائرة على CD تقوم بتنصيبه على حاسبك وهو يقوم بهذه المهمة في القليل جدا من الخطوات. وهو استخدام لوحات الأردوينو يمكنهم استخدام هذه اللوحة لحرق البرنامج على شريحة المتحكم، وتتفاصيل ذلك يمكن تنزيلها أيضا من على الإنترن特 واتباعها. نحن هنا سنفترض أن لدينا دائرة المسماة بدائرة برمجة المتحكم وهو في النظام In System Programmer, ISP في شكل ٤-٢٢. أيا كان نوع المبرمجة التي ستستعملها فإنها في النهاية ستتعامل مع أربع أطراف في شريحة المتحكم خاصة بالتوacial التتابعى الذى سنفرد له فصلا خاصا فيما بعد، ولكن ما يهمنا هنا هو كيفية توصيل المبرمجة مع هذه الشريحة. في العادة تأتى المبرمجة

مجهزة بكابل شريطي من ناحية التواصيل مع المتحكم وكابل USB من ناحية التواصيل مع الحاسب. الكابل الشريطي ينتهي بسوكت (قاعدة) ذات ٦ أو ١٠ أطراف كما في شكل ٤-٢٣ الذي يبين إسم كل طرف من أطراف هذه القاعدة. هذه الأسماء يقابلها نفس الأسماء تماما على شريحة المتحكم كما هو موضح في شكل ٤-٢٣-٤ أيضا. المطلوب منك هو توصيل الأطراف التي لها نفس الإسم في سوكت المبرمجة بالمقابل لها من أطراف المتحكم كما في شكل ٤-٢٤ الذي يوضح مصدر القدرة أيضا. بعد ذلك تقوم بتشغيل البرمجية

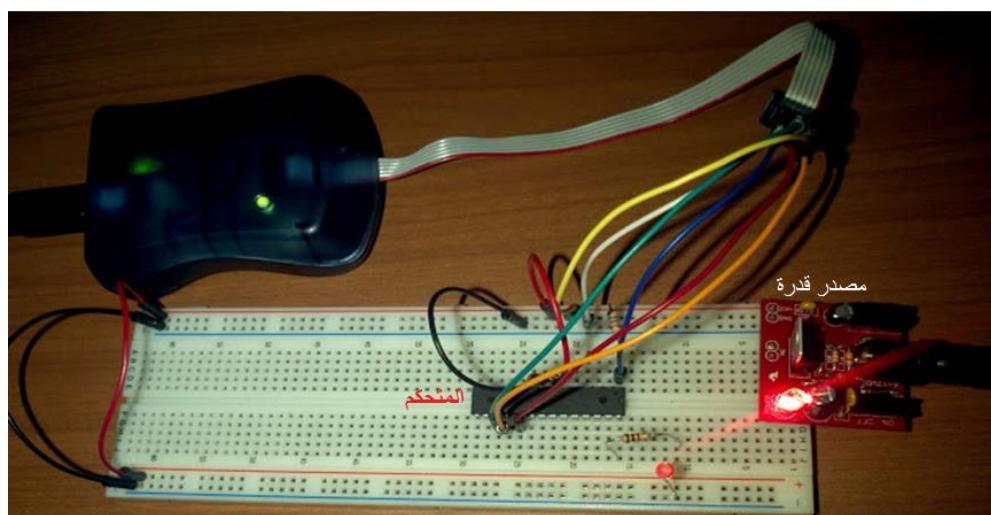
شكل ٤-٢٢ دائرة برمجة المتحكم (المبرمجة)  
وهو في النظام



الخاصة بالمبرمجة للبدأ في نقل الملف المستعشرى إلى ذاكرة المتحكم، ثم نقل المتحكم إلى مكانه في النظام المدمج أو الكامن. لقد قمنا في هذا التوضيح بنقل المتحكم على لوحة الاختبار من أجل برمجته، ولكن في المشاريع الكبيرة يتم تجهيز سوكت على لوحة النظام الكامن بحيث يتم تثبيت الكابل الشريطي القادم من المبرمجة فيها مباشرة، بحيث لا تكون مضطرا لنزع المتحكم من مكانه بل تتم عملية البرمجة وهو في مكانه. بذلك تكون قد انتهينا من دورة تطوير أى نظام مدمج أو كامن والتي نلخصها في الخطوات الثلاث التالية:



شكل ٤-٢٣ أشكال سوكيل المبرمج وأطراف المتحكم المقابلة لها



شكل ٤-٢٤ توصيل أطراف المبرمج على الأطراف المقابلة لها في المتحكم

١- كتابة برنامج المتحكم بأى لغة تفضلها (لغة التجميع الخاصة بالمحكم أو استخدام أحد اللغات ذات المستوى العالى)، ونحن فى هذا الكتاب سنستخدم لغة C++ لكثرة شيع استخدامها، كما أنها غالبا هى اللغة التى يتعلمنها الطلاب فى المراحل الأولى من كليات الهندسة فى معظم الجامعات. ولقد أشرنا مسبقاً أننا سنستخدم الأتمل استديو كوسط برمجة متكملاً لتحرير واختبار ومحاكاة البرنامج إلى أن نحصل على صورة من البرنامج صحيحة خالية من الأخطاء تؤدى المطلوب منها، وبالطبع سيتم الحصول على النسخة الستعشورية لهذا البرنامج.

- ٢ - بناء دائرة المشروع واختبارها ومحاكاتها على برنامج بروتوس إلى أن يتم الحصول على القيم المناسبة لكل المكونات.
- بالطبع سيتم استخدام النسخة المستعشرية للبرنامج التي يتم الحصول عليها من الخطوة الأولى. كل الدوائر أو المشاريع التي سيتم استخدامها في هذا الكتاب سيتم تفاصيلها واختبارها باستخدام هاتين الخطوتين فقط.
- ٣ - الخطوة الثالثة هي الحصول على النموذج الأولى للمشروع على لوحة مطبوعة (أو على لوحة اختبار في المشاريع البسيطة)، وهذه لن نطرق إليها بل نتركها للقاريء لوجود الكثير من الوسائل والبرمجيات المساعدة في ذلك، ونحن لا نريد التشعيّب إلى ذلك حتى نركز على فهم الإمكانيات المختلفة للمتحكمات والذي هو الهدف الأساسي من هذا الكتاب.

## ٤-٧ المشروع الثاني

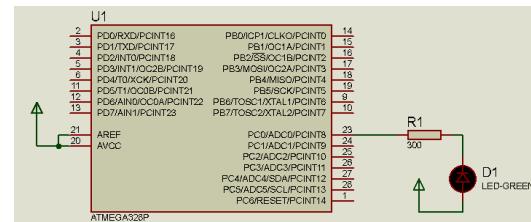
### إنارة وإطفاء دايوود ضوئي بمعدل معين

في المشروع الثاني ضمن موضوع إدخال وإخراج البيانات سنستخدم دايوود ضوئي واحد ويقوم البرنامج بإثارته وإطفاؤه بمعدل معين ول يكن نصف ثانية بين الإنارة والإطفاء. كل ما يحتاجه هذا البرنامج من مكونات هو دايوود ضوئي LED واحد ومقاومة ٣٠٠ Ω يتم توصيلهم على أحد أطراف أي واحدة من البوابات الثلاثة في المتحكم atmega328 ولتكن على الطرف رقم صفر من البوابة C، أي PC0. البرنامج الذي سيقوم بهذه المهمة سيكون كالتالي:

```

/*
* Led2.c
*
* Created: 6/10/2017 9:01:10 AM
* Author : M. Eladawy
*/
#define F_CPU 1000000 //CPU freq. in
hertz
#include <util/delay.h> //delay in util library
#include <avr/io.h>
int main(void)
{
    DDRC=0xFF ; // port C output

```



```
while (1)
{
    PORTC=0xff; // output ones on port C
    _delay_ms(500) ; //delay 500 millisecond
    PORTC=0x00 ; // zeros on port C
    _delay_ms(500) ; //delay half second and repeat
}
}
```

الأمر الأول #define F\_CPU 1000000 وهو واحد من الأوامر التوجيهية للمترجم الخاص بلغة C (compiler) التي تخبره بعمل أشياء معينة قبل البدأ في تنفيذ دالة البرنامج الأساسية main، وهذا الأمر الذي معنا يخبر المترجم بأن يعتبر أن تردد نبضات التزامن الخاصة بوحدة المعالجة المركزية CPU هي 1000000 هرتز، وهو في الحقيقة تردد مصدر النبضات الداخلي في المتحكم. لابد أن يكتب التردد بهذه الطريقة (واحد بجواره ٦ أصفار)، ويجب أن تكون بهذه القيمة بالضبط حتى تكون قيم أزمنة التأخير التي سيتم استخدامها داخل البرنامج صحيحة لأن أزمنة التأخير يتم حسابها عن طريق برمجيات صغيرة تحسب عدد معين من حلقات التأخير التي تعتمد على هذا التردد. يمكنك التتحقق من ذلك بوضع التردد يساوى 4000000 (أربعة بجوارها ٦ أصفار) مثلاً لترى أن معدل إضاءة وإطفاء الديايد الضوئي سيكون بطبيعاً جداً بالنسبة للقيمة 1000000، والعكس صحيح، فإنه بوضع هذا التردد يساوى 500000 (خمسة بجوارها ٥ أصفار) ستتجدد أن معدل الإضاءة والإطفاء سيكون أسرع بكثير. لذلك لابد من الالتزام بقيمة التردد 1000000 حتى نحصل على أزمنة تأخير صحيحة وبالقيم المكتوبة في أوامر التأخير التي ستنشرحها بعد قليل. الأمر الثاني #include <util/delay.h> يخبر المترجم بضم ملف التأخير delay.h الموجود في مكتبة الخدمات utility للبرنامج لأنه سيتم استخدام أوامر تأخير في البرنامج ستحتاج لهذا الملف. الأمر الثالث #include <avr/io.h> يخبر المترجم بضم ملف الإدخال والإخراج الموجود في مكتبة AVR. بعد ذلك تم الدخول في الدالة الأساسية للبرنامج main. الأمر الأول في الدالة main هو DDRC=0xFF الذي يجعل كل أطراف البوابة C أطراف إخراج. بعد ذلك تم الدخول في الحلقة اللاحائية (1) while وتنفيذ الأمر الأول فيها PORTC=0xff الذي يضع وحيداً على كل أطراف البوابة C، ثم يأتي بعد ذلك أمر التأخير (500) delay\_ms الذي ينادي على ملف التأخير كما ذكرنا والذي يتطلب منه تأخير مقداره ٥٠٠ ميلليثانية، والذي يساوى نصف ثانية يتم تحريرها ملف التأخير بين القوسين. لابد أن يكتب الأمر بهذه الطريقة. هناك أيضاً الأمر (delay\_us) الذي يعطى تأخيراً بلمائة روثانية. الأمرين التاليين يضعان أصفراً على خرج البوابة C لإطفاء الديايد ثم التأخير بمقدار نصف ثانية، ثم يتم تكرار ذلك من خلال الحلقة اللاحائية. بعد أن فهمنا البرنامج يمكنك اللعب بأزمنة التأخير المختلفة ورؤيه النتيجة في معدل إضاءة وإطفاء الديايد. تذكر أنه مع كل

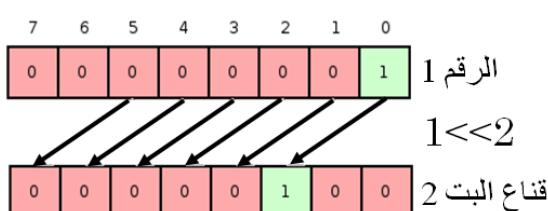
تغيير للتردد عليك أن تقوم بإعادة بناء البرنامج Build في الأتمل استديو، لتجديد القيمة في النسخة الستعرية للبرنامج، ثم الذهاب إلى برنامج البروتس وتحميل الملف الستعرى من جديد، ثم تنفيذ البرنامج لترى تأثير التغيير الذى أحدثته.

## ٤-٨ التعامل مع برات محددة داخل أي متغير

إفترض مثلاً أن لدينا المتغير  $a=11110000$ ، والمطلوب هو جعل البت رقم 1 تساوى 1 بدلًا من صفر (تذكر أنها نعد البتات بدءاً من البت رقم 0 وهى البت التي في أقصى اليمين)، أي أننا نريد جعل المتغير  $a$  يصبح  $a=11110010$ ، فكيف يتم ذلك؟ هذه المهمة تستخد بكثره جداً في برامج المتحكمات. إننا يمكن أن ننفذ ذلك بتنفيذ الأمر  $a=0b11110010$ ، أو باستخدام القناع والإزاحة كما سنرى هنا.

### القناع لأى بت

القناع mask للبت واحد هو  $00000010$ ، وهو رقم كله أصفار ما عدا البت رقم 1 توضع بالقيمة 1، بنفس الطريقة فإن قناع البت رقم 5 سيكون  $00100000$ ، أي أن كل البتات تساوى أصفاراً ما عدا البت رقم 5، وعلى ذلك فإن قناع أي بت رقمها  $n$  سيكون رقم كل باته أصفاراً ما عدا البت رقم  $n$ . الآن أنظر إلى الرقم الستعرى 1 الذي يمكن كتابته على الصورة  $0x01=00000001$ . بإجراء إزاحة ناحية اليسار بمقدار 2 بت على برات الرقم 1 يصبح الناتج كالتالى  $00000100$ . هذه العملية يمكن التعبير عنها في لغة C كالتالى:  $1 <> 2$ ، حيث الرمز  $<>$  يعني الإزاحة ناحية اليسار، والرقم على يساره هو الرقم المراد إزاحته، والرقم الذي على يمين الرمز هو مقدار الإزاحة بالبتات،



شكل ٢٦ الحصول على قناع البت ٢ من الرقم ١

وبالتالى فإن التعبير  $1 <> 2$  يعني إزاحة برات الرقم 1 ناحية اليسار بمقدار 2 بت. وعلى ذلك فإن قناع أي بت  $n$  يمكن الحصول عليه بإزاحة الرقم 1 ناحية اليسار بمقدار  $n$  من البتات باستخدام التعبير التالي  $n <> 1$ .

الآن لو أجرينا عملية الأول OR المنطقية على محتويات الرقم  $a=11110000$  مع قناع البت رقم 1 الذي نحصل عليه بإزاحة الرقم 1 بمقدار بت واحدة وهو  $00000010$ ، فإن الناتج سيكون  $11110010$  وهو نفس الرقم  $a$  مع تغير البت رقم 1 فقط من صفر إلى 1. يمكن التعبير عن ذلك بالأمر التالي:  $(2 <> 1) | a$  والذي يعني جعل الرقم

$a$  يساوى نفسه (a) بعد إجراء عملية أور على محتوياته مع ناتج إزاحة الرقم 1 بمقدار 2 بت. التعبير  $= a | a$  يتم نطقه كالتالي a تساوى a أورد مع. والتعبير  $= a | a$  ينطق كالتالي: a تساوى a أورد مع الرقم 1 المزاح بمقدار 2 بت. على ذلك فإن التعبير  $= a | a$  سيكون ناتجه  $a = 11110001$ ، والتعبير  $= a | a$  سيكون ناتجه  $a = 11111000$ ، والتعبير  $= a | a$  سيكون  $a = 11110000$  (أى لا تغيير) حيث 1 أور 1 هو 1.

### العمليات المختلفة على مستوى البت في لغة C

يبين الجدول 1 العمليات المنطقية المختلفة التي يمكن إجراؤها على اثنين بت ورمز كل منها.

**الجدول 1 العمليات المنطقية على مستوى البت**

رمز العملية	العملية
&	عملية الآند AND، تعطى 1 إذا كان الاثنين بت وحيد، وتعطى صفرًا فيما عدا ذلك
	عملية الأور OR، تعطى صفرًا إذا كان الاثنين بت يساويان صفرًا، وتعطى واحد فيما عدا ذلك
^	عملية الإكس أور XOR (هي الرمز الموجود فوق الرقم 6 على لوحة المفاتيح)، تعطى واحد إذا كان الاثنين بت مختلفين، وتعطى صفرًا إذا كان الاثنين بت متساوين
~	عملية العكس NOT، تعكس محتويات أي بت
<>	الإزاحة لليسار، البتات الخارجية من اليسار تهمل أو تضيع، ويتم إدخال أصفار من ناحية اليمين في البتات الزائدة
>>	الإزاحة ناحية اليمين، البتات الخارجية من اليمين تهمل أو تضيع، ويتم إدخال أصفاراً من ناحية اليسار في البتات الزائدة.

ما سبق يتضح أنه لجعل أي بت في أي رقم تساوى واحد، أي عمل setting لهذه البت، فإننا نجري عملية أور على رقم مع قناع mask هذه البت،  $(n <= a)$ . الآن ماذا نفعل لو أردنا تصفير任意 بت من بتات أي رقم بدون تغيير بقائه الآخر؟.

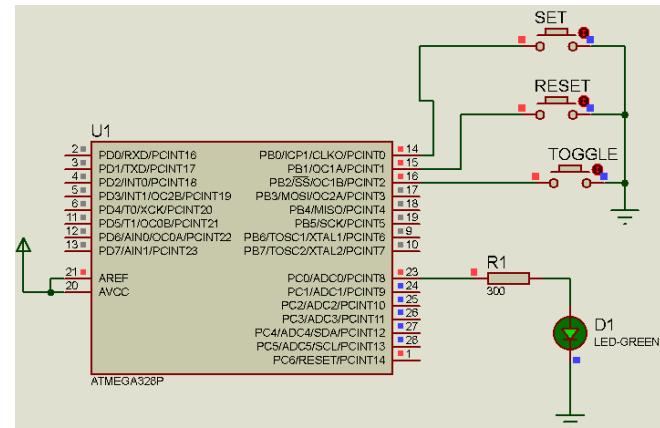
يتم ذلك بإجراء عملية آند على محتويات الرقم مع معكوس قناع هذه البت. إفترض أيضًا الرقم  $a = 11110000$  والمطلوب هو تصفير البت رقم 5. قناع البت رقم 5 هو 00100000، ومعكوس هذا القناع هو 11011111، وبإجراء عملية آند على هذا المعكوس مع الرقم a نحصل على الرقم a الجديد وهو  $a = 11010000$ ، بعد أن تم تصغير البت رقم 5 فيه. الأمر الذي سيقوم بهذه العملية هو  $= a \& \sim(1 <= 5)$ ، والذي ينطق a تساوى a بعد عمل آند لها مع معكوس قناع البت رقم 5.

العملية الأخيرة التي سرّاها هنا هي عملية عكس toggle أي بت من بtas أي رقم. إفترض أيضاً أن الرقم a=11110000، والمطلوب عكس كل من البت رقم ٢ والبت رقم ٥. قناع البت رقم ٢ هو ٠٠٠٠٠١٠٠، وبإجراء عملية إكس أور على محتويات الرقم a مع هذا القناع فإن الناتج سيكون ١١١١٠١٠٠ حيث نرى انعكاس البت رقم ٢. الأمر الذي سينفذ هذه العملية  $(a^{\wedge} 2) \lll 2$ ، ولكن عكس كل من البت رقم ٢ ورقم ٥، فإن الأمر لذلك سيكون  $((a^{\wedge} 5) | (a^{\wedge} 2)) \lll 5$  حيث سيكون الناتج هو ١١٠١٠١٠٠. المشروع التالي سيوضح التعامل بهذه الأوامر المختلفة.

## ٤-٩ المشروع الثالث

### التحكم في دايد ضوئي من خلال ثلاث مفاتيح

سنحاول في هذا المشروع التدريب على التعامل مع بت بعينها في أحد المتغيرات من خلال الإزاحة. سنجعل البوابة C بوابة خرج مع وضع دايد ضوئي ومقاومة على المخرج الأول منها PC0 بحيث يضيء عند إخراج واحد على هذا الطرف. في المقابل سنجعل البوابة B بوابة دخل مع وضع ثلث مفاتيح على المداخل الثلاثة الأولى فيها، PB0 و PB1 و PB2. المطلوب عند الضغط على المفتاح الأول أن يضيء الدايد الضوئي، وإن كان مضينا في الأصل، يبقى كما هو، ولذلك سنسميه مفتاح الوضع SET. المفتاح الثاني عند ضغطه يطفئ الدايد الضوئي، وإن كان مطفأً في الأصل يظل كما هو، ولذلك سنسميه مفتاح إعادة الوضع RESET. المفتاح الثالث عند الضغط عليه نريده أن يغير وضع الدايد الضوئي، فإن كان مطفأً يضيء، وإن كان مضينا يطفئه، ولذلك سنسميه مفتاح تغيير الوضع TOGGLE.



شكل ٤٢٧-٤ تنفيذ مشروع Led3  
فإن كان مطفأً يضيء، وإن كان مضينا يطفئه، ولذلك سنسميه مفتاح تغيير الوضع TOGGLE.  
كالتالي، وتنفيذته على البروتوس سيكون كما في شكل ٤٢٧-٤.

/\*

\* Led3.c

\*

\* Created: 6/11/2017 2:15:15 PM

\* Author : M. Eladawy

\*/

```
#include <avr/io.h>
int main(void)
{
    DDRB=0x00; //port B input
    DDRC=0xFF; //port C output
    PORTB |= (1 << PORTB0)|(1 << PORTB1)|(1 << PORTB2);
    //Turn on pull up resistance
    on the first three inputs
    while (1)
    {
        if ((PINB & (1<<0))==0)
            {PORTC |= (1<<0);}
        //Setting PC0
        if ((PINB & (1<<1))==0)
            {PORTC &= ~ (1<<0);} //Resetting PC0
        if ((PINB & (1<<2))==0)
            {PORTC ^= (1<<0);} //Toggle PC0
    }
}
```

في هذا البرنامج أنظر إلى الأمر :

PORTB |= (1 << PORTB0)|(1 << PORTB1)|(1 << PORTB2);

هذا الأمر يستخدم طريقة القناع والإزاحة لوضع وحاید على المخارج الثلاثة الأولى من البوابة B. مثلاً التعبير  $<<1$  يجعل قناع الرقم 1 وهو 00000001 يزاح بمقدار الرقم PORTB0 وهو صفر، لذلك سيكون الناتج هو نفس القناع، والتعبير  $<<1$  سيزاح قناع الرقم 1 بمقدار الرقم PORTB1 وهو 1، أي أن هذا التعبير يكافئ التعبير  $<<1$ ، ولذلك سيكون الناتج هو 00000010، وبنفس الطريقة ستكون نتيجة التعبير  $<<1$  هى 00000100. بعد ذلك سيقوم هذا الأمر بعمل أور ل هذه التعبيرات الثلاثة التي ستكون

نتيجة لها هي 00000111 حيث يقوم بإخراج هذا الرقم على أطراف البوابة B. السؤال المهم هنا والذى يجب أن نذكره جيدا هو لماذا نخرج ثلث وحايد على الأطراف الثلاثة الأولى للبوابة B على الرغم من أن البوابة B تم تحديدها أصلا لتكون بوابة إدخال. بالرجوع إلى شكل ٤-٤ سنتذكر أن إخراج 1 على طرف يتم تعينه كطرف دخل في أي بوابة لا يخرج هذا الواحد على طرف البوابة، ولكن هذا الواحد ينشط مقاومة الجذب pull up الداخلية لهذا الطرف. لذلك فإن هذا الأمر سيقوم بتنشيط مقاومة الجذب pull up الداخلية على هذه الأطراف الثلاثة. تذكر أنه عند توصيل أي مفتاح على أي طرف دخل لابد من استخدام مقاومة جذب كما رأينا قبل ذلك في شكل ٤-٤. في هذا الشكل تم استخدام مقاومة جذب خارجية كما رأينا. هنا، وفي هذا البرنامج، باستخدام هذا الأمر لإخراج 1 على أي طرف دخل، سيعمل على تنشيط مقاومة الجذب الداخلية الموجودة على هذا الطرف وبالتالي يتم الاستغناء عن توصيل مقاومات جذب خارجية كما رأينا في الشكل ٤-٢٧، حيث تم توصيل المفاتيح بدون مقاومة الجذب الخارجية.

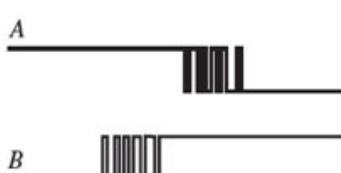
بعد ذلك دخل البرنامج في الحلقة اللا杭ائية التي تحتوى على ثلاثة أوامر شرطية if statements، حيث الأمر الشرطي الأول يحتوى التعبير التالي:  $(PINB == 0) & (PORTC == 1)$ ، يقوم بإجراء عملية آند على محتويات البوابة B مع الرقم 00000001 (وهو ناتج التعبير  $0 <> 1$ ) ويسأل إذا كان هذا الناتج يساوى 0 أم لا. معنى ذلك أن عملية الآند تساوى صفر يعني أن المفتاح SET في شكل ٤-٢٧ تم ضغطه مما جعل الطرف 0 = PB0. على ذلك فإذا كان نتيجة هذا الشرط حقيقة (ناتج الآند يساوى صفر فعلا) فإنه يتم جعل الخرج PC0 يساوى واحد باستخدام الأمر التالي للشرط وهو  $\{ (PORTC == 0) | (PINB == 1) \}$ . يأتي بعد ذلك أمر الشرط التالي وهو  $(PINB == 0) & (PORTC == 1)$  الذي يقوم بإجراء عملية آند على محتويات البوابة B مع الرقم 00000010 وهو ناتج التعبير  $1 <> 1$ ، ليり إذا كان المفتاح الثاني RESET تم ضغطه أم لا، بحيث إذا كان مضغوط يتم تصفير الطرف 0 PC0 وبالتالي إطفاء الدايدود الضوئي. بنفس الطريقة يتم الشرط الثالث لنرى إذا كان المفتاح الثالث مضغوطا أم لا، فإذا كان مضغوط يتم تغيير حالة الدايدود الضوئي. حاول تنفيذ هذا المشروع وللعبة بالمفاتيح الثلاثة لنرى مدى صحة البرنامج.

بتتنفيذ البرنامج Led3 كما في شكل ٤-٢٧ وللعبة بالمفاتيح الثلاثة سنجد أن كل من المفتاح الأول SET، والمفتاح الثاني RESET بالذات يعملان بصورة جيدة، أما المفتاح الثالث فلا يعمل بصورة جيدة، بل تكون نتيجته غير متوقعة، نتيجة التأثير الاهتزازي للمفتاح. عند الضغط على أي مفتاح ميكانيكي لتغير حالته من واحد إلى صفر مثلا، فإن ما يحدث هو تأرجح للمفتاح بين الصفر والواحد لفترة زمنية صغيرة جدا بعد الضغط على المفتاح. ولذلك سيكون الوضع النهائي الذي سيقف عنده المفتاح غير متوقع، حاول اللعب بهذا المفتاح بالذات وهو مفتاح TOGGLE في شكل ٤-٢٧ الذي تظهر فيه هذه المشكلة بوضوح. لماذا لم تظهر هذه المشكلة مع كل من المفتاحين الأول والثانى؟

ستترك الإجابة على هذا السؤال للقارى ليفكر فيها. يبقى السؤال المهم هنا وهو كيف نتخلص من تأثير هذه الاهتزازات في المفاتيح الميكانيكية؟

## ٤-١٠ الاهتزازات في المفاتيح الميكانيكية

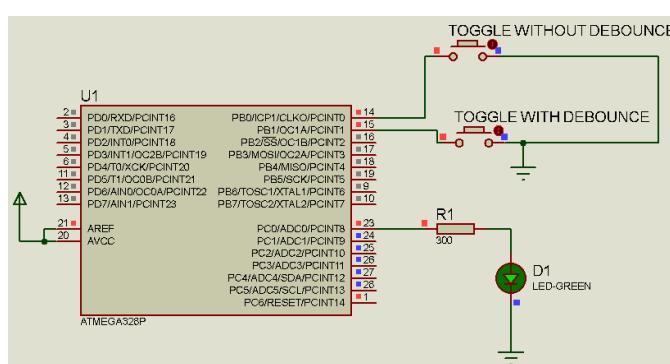
عند الضغط على أي مفتاح ميكانيكي ومن، أمثلتها مفاتيح لوحة المفاتيح، لتغيير حالته من صفر إلى واحد أو من واحد إلى صفر، فإن هذا التغيير لا يتم في نقلة واحدة، ولكن ما يحدث هو اهتزاز ميكانيكي للمفتاح عند لحظة التغيير مما ينتج عنه العديد من النبضات كما في شكل ٢٨-٤ ، وبالتالي سيكون التأثير على الخرج غير مرغوب فيه وربما يكون



شكل ٢٨-٤ الاهتزازات الميكانيكية الناتجة عن تغيير وضع أي مفتاح

غير متوقع لذلك يجب دائماً التخلص من تأثير هذه الاهتزازات. يتم التخلص من تأثير هذه الاهتزازات بطرقتين، إما باستخدام مكونات إلكترونية hardware، أو بطريقة برمجية software. التخلص من هذه الاهتزازات باستخدام مكونات إضافية سيحتاج بالطبع لإضافة مكونات قد تعقد الدائرة، كما أنها ستحتاج للضبط الدقيق لقيم هذه المكونات المضافة وبالذات المقاومات والمكثفات التي يتم استخدامها في الغالب. لذلك تستخدم هذه الطريقة مع الدوائر التي لا تحتوى على

أحد الأجهزة القابلة للبرمجة مثل المعالجات أو المتحكمات. أما إذا كانت الدائرة تحتوى متحكم أو معالج فإنه يمكن في هذه الحالة استخدام طريقة البرمجيات software للتخلص من تأثير هذه الاهتزازات. لكن فهم هذه الطريقة سفترض الدائرة البسيطة الموضحة في شكل ٢٩-٤ . في هذه الدائرة تم تنشيط مقاومة الجذب الداخلية على أول طرفين من البوابة B كما سبق. البرنامج



شكل ٢٩-٤ التخلص من الاهتزازات بالتأخير الزمني

التالى يحتوى تشغيل الدائرة بحيث أن المفتاح الأول لا يهتم بالتخلص من الاهتزازات، أما المفتاح الثانى فقد تم تعديل طريقة قراءته بحيث تخلصنا من هذه الاهتزازات. ما حدث هنا هو أنه عند قراءة المفتاح الثانى بالأمر &((PINB ==0))>><1>, والسؤال إذا كان صفر أم لا، فإنه يتم بعد ذلك عمل تأخير مقداره ٢٥ ميلليثانوية بالأمر،

، ثم قراءة نفس المفتاح مرة ثانية، فإذا كان المفتاح لا زال على وضعه ويساوي صفر، فإن ذلك يعني أن مرحلة الاهتزاز قد مضت واستقر المفتاح على الصفر، وفي هذه الحالة يتم عمل تغيير للطرف PC1، أما إذا كان المفتاح يساوى 1، بعد مرور ٢٥ ميللى ثانية فإن ذلك يعني أن هذا الواحد ناتج من الاهتزاز وفي هذه الحالة لا يتم عمل أى إجراء، ولكن يتم إعادة الخطوات السابقة.

```
/*
* Led4.c
*
* Created: 6/13/2017 8:51:54 AM
* Author : M. Eladawy
*/
#include <avr/io.h>
#define F_CPU 1000000
#include <util/delay.h>
int main(void)
{
    DDRB=0x00; //port B input
    DDRC=0xFF; //port C output
    PORTB |= (1 << PORTB0)|(1<<PORTB1);
    //activation of pull up resistance on PB0 and PB1
    while (1)
    {
        if ((PINB & (1<<0))==0)
            {PORTC ^= (1<<0);} //Toggle PC0
        if ((PINB & (1<<PINB1))==0)
            {_delay_ms(25);
            if( (PINB & (1<<PINB1)) == 0)
                {PORTC ^= (1<<0);} //Toggle PC0
            else { }
            }
    }
}
```

{

حاول كتابة هذا البرنامج وللعبة بالمفاتيح لترى تأثير الاهتزازات وكيفية التخلص منها عن طريق التأخير الزمني من خلال البرنامج .software

## ملخص الفصل

بدأ الفصل باستعراض خصائص المتحكم atmega328 مع التركيز على أن هذه الخصائص متوافقة تماماً مع الخصائص العامة لمتحكمات AVR التي تم شرحها في الفصل ٣. بعد ذلك تم استعراض وسط البرمجة أتمل استديو كوسط لكتابه البرامج بلغة C ومحاكاتها وتصحيحها debugging، وتم ذلك من خلال كتابة برامج لإدخال وإخراج بيانات من البوابات المختلفة للمتحكم، وذلك طبعاً بعد التعرف على كيفية تشغيل هذه البوابات للقيام بعمليتي إدخال أو إخراج البيانات على كل خط من خطوطها، والتراكيب الداخلي، أو الإلكترونيات، الخاصة بهذه البوابات. كل هذه البرامج تمت محاكاتها على برنامج الأتمل استديو والتأكد من صحتها، ثم تم نقلها وتضمينها، أو حرقها، على شريحة المتحكم في برنامج البروتوس من أجل محاكاة تشغيلها مع الدوائر الموصولة عليها من الخارج. بإتمام هذا الفصل مفروض أن يستطيع القارئ التعامل مع البوابات الثلاثة في المتحكم من أجل إدخال أو إخراج البيانات مع التعرف أيضاً على كيفية الحصول على أزمنة التأخير.

## الفصل ٥

تطبيقات على إدخال وإخراج البيانات

Application on Data Input/ Output

**العناوين المضيئة في هذا الفصل:**

- ١- التعامل مع شاشات العرض ذات البلاوره السائلة
- ٢- إنشاء مكتبة للتعامل مع الشاشات LCD
- ٣- التعامل مع مصفوفة مفاتيح

## ١-٥ مقدمة

**سنقوم** في هذا الفصل بالشرح التفصيلي لبعض التطبيقات على إدخال وإخراج البيانات من وإلى المتحكم atmega328 والتي لا تتطلب التعامل مع أي ملحقات أخرى في المتحكم لم يتم دراستها حتى الآن، حيث سيكون هناك تطبيقات على كل واحد من هذه الملحقات في الفصل الخاص به.

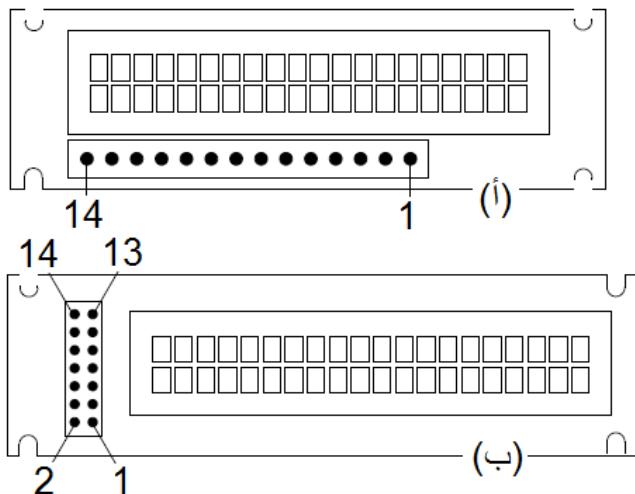
## ٢-٥ التعامل مع شاشات العرض البليورية LCD

في العادة تستخدم شاشات العرض البليورية LCDs في الكثير من تطبيقات النظم الكامنة embedded systems التي تستخدم المتحكمات. من مميزات هذه الشاشات أنها صغيرة الحجم، ذات تكلفة معقولة، وسهولة تثبيتها في أي مكان في النظام الكامن أو المدمج. توجد هذه الشاشات في نوعين، نوع خاص بعرض الحروف اللغوية، وتسمى الشاشات الألجدية alphanumeric، وبالطبع المقصود هنا هو الحروف الإنجليزية، والنوع الثاني هو الشاشات الرسومية Graphical والتي يمكن بواسطتها عرض رسومات معينة يمكن من خلالها عرض الأحرف العربية. سنعرض في هذا الفصل لنوع الأول الخاص بعرض الحروف الهجائية فقط. توجد شاشات العرض الألجدية في أنواع كثيرة من حيث طول السطر، أو عدد الأحرف في السطر الواحد، حيث توجد وحدات بها ٨ و ١٦ و ٢٠ و ٢٤ و ٣٢ و ٤٠ حرفاً في السطر الواحد. وتوجد هذه الشاشات بها أكثر من سطر حيث توجد شاشات بها سطر واحد، وسطران، وثلاثة أسطر، وأربعة أسطر، وأكثر من ذلك.



شكل ١-٥ أشكال مختلفة لشاشات العرض البليورية LCD

شكل ٥-١ يبين صورا مختلفة لأنواع المختلفة من هذه الشاشات ويمكنك التعرف في الشكل على الشاشات الأبجدية والرسومية.



شكل ٢-٥ سوكيل توصيل الشاشات وطريقة ترقيمها

بالنسبة لسوكيت التوصيل لهذه الشاشات فهي توجد في نوعين وكلا النوعين يتكون من ١٤ نقطة توصيل قد تصل إلى ١٦ نقطة في بعض أنواع الشاشات. النوع الأول يحتوى كل النقاط مرصوصة في صف واحد، والنوع الثاني يحتوى هذه النقاط موضوعة في صفين. شكل ٢-٥ يبين هذين النوعين وطريقة ترقيم هذه النقاط.

جدول ١-٥ وظائف الأطراف المختلفة لشاشة العرض البليوروبية

رقم الطرف	رمزه	وظيفته
١	Vss	الأرضي
٢	Vcc	جهد مصدر القدرة، ٥ فولت
٣	Vee	جهد التحكم في التباين contrast
٤	RS	إختيار مسجل البيانات أو مسجل الأوامر، Register Select = مسجل الأوامر، ١ = مسجل البيانات
٥	R/W	القراءة أو الكتابة، ٠ = كتابة، ١ = قراءة من الشاشة
٦	E	طرف تنشيط enable عالي الفعالية، أي يوضع بواحد ليتم التنشيط
٧	DB0	الطرف ٠ في مسار البيانات
٨	DB1	الطرف ١ في مسار البيانات
٩	DB2	الطرف ٢ في مسار البيانات
١٠	DB3	الطرف ٣ في مسار البيانات
١١	DB4	الطرف ٤ في مسار البيانات
١٢	DB5	الطرف ٥ في مسار البيانات
١٣	DB6	الطرف ٦ في مسار البيانات
١٤	DB7	الطرف ٧ في مسار البيانات
١٥	A	الأئود أو طرف الموجب للتحكم في شدة الإضاءة الخلفية (اختياري)
١٦	K	الكتود أو طرف السالب للتحكم في شدة إضاءة الخلفية (اختياري)

لاحظ أنه في الأنواع ذات الصف الواحد يكون ترقيم النقاط من اليمين لليسار، بينما في النوع ذو الصفين فإن الترقيم يكون من الأسفل يميناً، حيث توجد النقطة رقم واحد، ثم على يسارها النقطة ٢، ثم النقطة ٣ فوق النقطة ١ والنقطة

## النصف الأيسر من شفرة الرقم

	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
0x0	ØøP~P							-	۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰					
0x1	!۱AQa۹						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							
0x2	"۲BZRbr						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							
0x3	#۳CScs						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							
0x4	\$۴DTdt						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							
0x5	%۵EUeu						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							
0x6	&۶FUVfv						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							
0x7	'۷GWgW						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							
0x8	(۸HXhx						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							
0x9	)۹IYiy						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							
0xa	*:JZjz						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							
0xb	+;KCKk<						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							
0xc	,<L¥1।						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							
0xd	-=M]m)						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							
0xe	.>N^n>						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							
0xf	/?O_o<						۹۸۷۶۵۴۳۲۱۰	۹۸۷۶۵۴۳۲۱۰							

شكل ٣-٥ الشفرة الستعشرية hexadecimal لـ حرف لكـ

ومن الممكن التعامل معها أيضاً من خلال ٤ خطوط فقط كما سنرى لتوفير الوصلات بين الشاشة والمتحكم. الأطراف ٤ و ٥ و ٦ هى أطراف تحكم في الشاشة وهى توضح الغرض من البيانات الموضوعة على مسار البيانات DB0 حتى DB7. فمثلاً الطرف RS هو اختيار لاعتبار البيانات التي على مسار البيانات تمثل أمر command يوضع في مسجل DB7 الأوامر، أم بيانات يراد عرضها ستوضع في مسجل البيانات، حيث تحتوى الشاشة على مسجلين أحدهما للأوامر والآخر للبيانات Data Register. فمثلاً عندما يكون الطرف RS=0 فإن ذلك يعني القراءة Command Register أو الكتابة من مسجل الأوامر على حسب حالة الطرف R/W، وعندما يكون الطرف RS=1، فإن ذلك يعني القراءة

## ٤. على يسارها كما في شكل ٢-٥.

جدول ١-٥ يبين إسم كل طرف من هذه الأطراف ووظيفته ورمزه. الطران ١٥ والأختياران، بمعنى أنهما قد يكونا موجودين في بعض الشاشات وغير موجودين في البعض الآخر. الطران ١٥ يتم توصيله على مصدر جهد ٥ فولت لإضاءةخلفية الشاشة، والطرف ١٦ يكون أرضي. الطرف ٣ الخاص بالتحكم في التباين يتم توصيله على جهد متغير من صفر إلى ٥ فولت، مع ملاحظة أن الجهد الأقل يعطى تبايناً أفضل، أي أن التنااسب تكون عكسية هنا. يجب ألا يزيد فرق الجهد على هذا الطرف عن جهد القدرة وهو  $V_{CC}$ .

من الواضح من جدول ١-٥ أن التعامل مع هذه الشاشات سيكون من خلال مسار بيانات مكون من ٨ بت (أطراف)،

أو الكتابة في مسجل البيانات على حسب حالة الطرف R/W. وعلى ذلك فإن خط التحكم R/W يوضح هل تريد القراءة أو الكتابة من هذين المسجلين. وأخيراً فإن خط التنشيط E يتم تنشيطه بعد تثبيت قيم خطوط التحكم والبيانات السابقة للبدأ في تنفيذ العملية المطلوبة كما سنرى عند التشغيل.

### الشفرات المستعشرية للأحرف المراد عرضها على الشاشة

عرض أي حرف على الشاشة يجب إرسال شفرة هذا الحرف المكونة من 8 بت إلى مسجل البيانات. شكل ٣-٥ يوضح الشفرة المقابلة لكل من الأحرف الممكن التعامل معها في معظم هذه الشاشات، حيث تم وضعها في صورة جدول مكون من ١٦ صفًا و ١٦ عمودًا. لقد تم ترتيب هذا الجدول بحيث أن أرقام الصفوف هي الأربع بنايات الأولى (الأربعة بنايات اليمنى، أو ذات القيمة المنخفضة) من شفرة الحرف معبرا عنها بالنظام المستعشرى hexadecimal، ولذلك فهي تتراوح من الصفر رقم صفر ٠x0 حتى الصفر رقم ١٥، أو ٠xf. أما أعمدة الجدول فأرقامها يمثلها الأربع بنايات الثانية (الأربع بنايات اليسرى، أو ذات القيمة العليا) من شفرة الحرف معبرا عنها بالنظام المستعشرى أيضاً، ولذلك فهي تتراوح من العمود رقم صفر ٠x0 حتى العمود رقم ١٥ أو ٠x0f. مثلاً الحرف A موجود في الصفر رقم ٠x1 والعمود ٤ وبالتالي ستكون شفرته المستعشرية هي ٠x41 والتي تقابل الشفرة ٠b01000001 الثنائية. بنفس الطريقة فإن الحرف N موجود في تقاطع الصفر رقم ٠xe والعمود رقم ٠x4، وبالتالي فإن شفرته المستعشرية ستكون ٠x4e، وشفرته الثنائية ستكون ٠b01001110، وهكذا يمكن الحصول على شفرة أي حرف أو أي شكل من الأشكال الموجودة في الجدول الموجود في شكل ٣-٥.

بعد أن عرفنا شفرة كل حرف يراد عرضه على الشاشة فما هي مجموعة الأوامر التي سنسخدمها في التعامل مع الشاشة.

### مجموعة أوامر الشاشة

شكل ٤ يبين جدولًا بمجموعة أوامر الشاشة. وهذه الأوامر كالتالي:

**١- مسح الشاشة Clear display:** وهذا الأمر يكتب حرف أبيض (فاضي) في جميع أماكن الشاشة. لاحظ من شكل ٣-٥ أن شفرة الحرف الفاضي هي ٠x20، لذلك فإن هذا الأمر سيكتب هذه الشفرة في جميع أحرف الشاشة. الشفرة الثنائية لهذا الأمر كما في شكل ٤-٥ هي ٠00000001 أو ٠x01. هذا الأمر يضع دليل الكتابة cursor في أعلى يسار الشاشة.

**٢- العودة لنقطة الأصل return home:** يضع دليل الكتابة في أعلى يسار الشاشة. شفرة هذا الأمر هي 0x02 أو 0x03 حيث أن العلامة # في البت B0 تعني لا يهم أن تكون هذه البت صفر أو واحد.

**٣- طريقة الإدخال entry mode set :** شفرة هذا الأمر تحدد حركة دليل الكتابة إذا كانت متزايدة، أي من اليسار لليمين إذا كانت البت (1/D) B1 تساوى 1، بينما إذا كانت هذه البت تساوى صفر فإن دليل الكتابة سيتحرك متناقصاً، أي من اليمين لليسار. بوضع واحد في البت B0 يتم تنشيط الإزاحة لدليل الكتابة، بينما عندما تكون هذه البت تساوى صفر، فإن الإزاحة تتوقف. هذا الأمر سيأخذ الشفرات من 0x04 حتى 0x07 على حسب حالة البت B0 والبت B1.

Command	Binary								Hex
	B7	B6	B5	B4	B3	B2	B1	B0	
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	1/D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x	10 to 1F
Function Set	0	0	1	8/4	2/1	10/7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

شكل ٤ - الأوامر المختلفة للتعامل مع الشاشة

**٤- إضاءة أو إطفاء الشاشة ودليل الكتابة display and cursor ON/OFF:** البت B2 خاصة بالشاشة ولذلك ستجد فيها حرف D، وبوضع هذه البت بواحد تضيء الشاشة وبوضعها بصفر تطفئ display الشاشة. البت B1 خاصة بدليل الكتابة cursor، ولذلك ستجد فيها الحرف c، عندما تكون هذه البت بواحد يتم تشغيل دليل الكتابة، وعندما تكون هذه البت بصفر يتم إطفاء دليل الكتابة. البت B0 خاصة بإطفاء وإضاءة blinkingحرف الذي يقف عنده دليل الكتابة بحيث عندما تكون هذه البت تساوى واحد فإن الحرف الذي يقف عنده دليل الكتابة سيتردد (إضاءة وإطفاء)، وعندما تكون هذه البت تساوى صفر فلن يحدث هذا التردد.

**٥- إزاحة دليل الكتابة أو دليل الكتابة والشاشة يميناً أو يساراً cursor/display shift:** البت B3 عندما تكون بصفر يتم إزاحة دليل الكتابة فقط، وعندما تكون بواحد يتم إزاحة كل من الشاشة ودليل الكتابة. اتجاه هذه الإزاحة يتحدد بالبت B2، فإذا كانت بصفر فإن الإزاحة تكون لليسار، وإذا كانت بواحد فإن الإزاحة تكون لليمين. لاحظ أن البت B1 والبت B0 لا يهم أن يكونا بصفر أو واحد.

**٦- ضبط الأداء function set:** البت B4 خاصة بتشغيل الشاشة من خلال مسار بيانات ٨ بت أو ٤ بت. عندما تكون 0=B4 فإن ذلك يعني أن التشغيل سيكون على أساس ٤ بت، وعندما تكون 1=B4 فإن التشغيل سيكون على أساس ٨ بت. البت B3 خاصة بحالة الشاشة هل هي سطر واحد أو سطرين، فعندما تكون 0=B3 فإن التشغيل سيكون على أساس سطر واحد، وإذا كانت 1=B3 فإن التشغيل سيكون على أساس سطرين. البت B2 تعرف تحديدية الخط، أو بمعنى آخر كثافة النقاط المكون منها كل حرف، فعندما 0=B2 فإن ذلك يعني أن كثافة النقاط ستكون 5x7 نقطة، وعندما 1=B2 فإن ذلك يعني أن كثافة النقاط ستكون 10x5 نقطة لكل حرف. لاحظ أيضاً أن البت B1 و B0 لا تم حالاتها أن تكون صفر أو واحد.

**٧- تحديد عنوان الذاكرة CGRAM:** تحديد عنوان الحرف في ذاكرة توليد الأحرف character generation RAM وهذه الذاكرة تتكون من ٦٤ بايت حيث هناك ٦ برات فقط مخصصة لتحديد هذا العنوان، ولذلك فإن هذه العناوين تتراوح من 0x40 حتى 0x7f كما في شكل ٤-٥. بمجرد تحديد عنوان بهذا الأمر فإنه يمكن بعد ذلك القراءة أو الكتابة في هذا العنوان.

**٨- تحديد عنوان في ذاكرة بيانات الشاشة DDRAM:** ذاكرة بيانات الشاشة display data RAM تحتوى على ١٢٨ عنواناً يمكن القراءة منها أو الكتابة فيها بعد تحديد العنوان بهذا الأمر. لذلك فإن هذه الأمر به ٧ بت مخصصة لهذا العنوان، والعنوان سيتراوح ما بين 0x80 و 0xff كما في شكل ٤-٥.

## التعامل مع الشاشة من خلال المتحكم atmega328

شكل ٥-٥ يبين دائرة المحاكاة من برنامج بروتوكس والتي سنتعامل معها في هذا الجزء. نظرة بسيطة على أطراف الشاشة سنجد أنها مقسمة إلى ثلاث مجموعات، الأولى هي خطوط البيانات D0 إلى D7 وهذه ستحتاج لها بوابة كاملة ولتكن البوابة B بعد جعلها بوابة إخراج لأننا سنخرج عليها البيانات المطلوب كتابتها على مسار بيانات الشاشة. بالطبع من الممكن استخدام أربع خطوط فقط على حسب وضع البت B4 في الأمر رقم ٦ من مجموعة الأوامر السابقة التي تختار مسار البيانات ليكون ٨ أو ٤ برات. المجموعة الثانية من الخطوط هي خطوط التحكم، وهي مكونة من الخطوط الثلاثة RS و R/W ولقد رأينا وظيفة كل منها من قبل، وهذه ستستخدم من خطوط أي بوابة أخرى ولتكن البوابة C بعد جعل خطوطها الثلاثة الأولى على الأقل كخطوط إخراج. مجموعة الخطوط الثالثة وعددتها ثلاثة خطوط وهي خطوط قدرة خاصة بالأرضي و Vcc والتحكم في التباين كما في شكل ٥-٥. البرنامج التالي سيكتب العبارة MOHAMED ELADAWY على الشاشة بحيث تكون الكلمة الأولى في السطر الأول والثانية في السطر الثاني كما في الشكل.

من الملاحظات العامة على البرنامج أنه بعد إعطاء أي أمر أو بيانات، لابد من تنشيط طرف التنشيط للشاشة وهو الطرف E بوضعه يساوى واحد ثم تنزيله للصفر والانتظار لفترة زمنية معينة (نحن وضعناها ثانية كاملة حتى نرى الكتابة حرف بحرف). أي أن التنشيط يكون في صورة نبضة (010) ثم الانتظار لفترة زمنية معينة. راجع البرنامج باطمئنان وحاول مراجعة الشفرات مع محتويات الشكلين ٣ و ٤ والتأكد من تطابقها.

```
/*
* LCD1.c
*
* Created: 6/17/2017 8:54:46 AM
* Author : Mohamed Eladawy
*/
#include <avr/io.h>
#define F_CPU 1000000
#include <util/delay.h>
int main(void)
{
    DDRB=0xFF; //PB output
    DDRC=0xFF; // PC output
    PORTC=0x00; //RS=0, command mode, R/W=0, write mode
    PORTB= 0x38; //set display 8 bit, 2 lines mode
    PORTC |=(1<<PORTC2); // enable
    PORTC &=~(1<<PORTC2); // enable
    _delay_ms(1000);

    PORTB= 0x01; //clear display
    PORTC |=(1<<PORTC2);
    PORTC &=~(1<<PORTC2);
    _delay_ms(1000);

    PORTB= 0x0f; // cursor ON
    PORTC |=(1<<PORTC2);
    PORTC &=~(1<<PORTC2);
    _delay_ms(1000);
```

```
POR TB=0x80; //seek 00, start of line 1
```

```
POR TC |=(1<<POR TC2);
```

```
POR TC &=~(1<<POR TC2);
```

```
_delay_ms(500);
```

```
POR TC |=(1<<POR TC0); // RS=1 data entry mode
```

```
POR TB=0x4d; //letter M
```

```
POR TC |=(1<<POR TC2);
```

```
POR TC &=~(1<<POR TC2);
```

```
_delay_ms(1000);
```

```
POR TB=0x4f; //letter O
```

```
POR TC |=(1<<POR TC2);
```

```
POR TC &=~(1<<POR TC2);
```

```
_delay_ms(1000);
```

```
POR TB=0x48; //letter H
```

```
POR TC |=(1<<POR TC2);
```

```
POR TC &=~(1<<POR TC2);
```

```
_delay_ms(1000);
```

```
POR TB=0x41; //letter A
```

```
POR TC |=(1<<POR TC2);
```

```
POR TC &=~(1<<POR TC2);
```

```
_delay_ms(1000);
```

```
POR TB=0x4d; //letter M
```

```
POR TC |=(1<<POR TC2);
```

```
POR TC &=~(1<<POR TC2);
```

```
_delay_ms(1000);
```

```
POR TB=0x45; //letter E
```

```
POR TC |=(1<<POR TC2);
```

```
POR TC &=~(1<<POR TC2);
```

```
_delay_ms(1000);
```

```
PORTB=0x44; //letter D
```

```
PORTC |=(1<<PORTC2);
```

```
PORTC &=~(1<<PORTC2);
```

```
_delay_ms(1000);
```

```
PORTC=0x00; // RS=0, command mode
```

```
PORTB=0xC0; //seek 00, start of line 2
```

```
PORTC |=(1<<PORTC2);
```

```
PORTC &=~(1<<PORTC2);
```

```
_delay_ms(500);
```

```
PORTC |=(1<<PORTC0); // RS=1 data entry mode
```

```
PORTB=0x45; //letter E
```

```
PORTC |=(1<<PORTC^
```

```
PORTC &=~(1<<PORTC)
```

```
_delay_ms(1000);
```

```
PORTB=0x4C; //letter L
```

```
PORTC |=(1<<PORTC)
```

```
PORTC &=~(1<<PORTC)
```

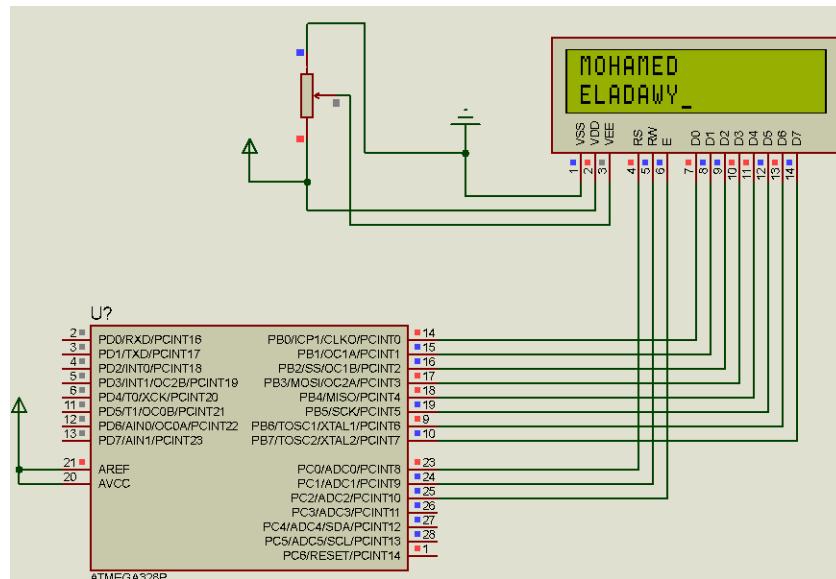
```
_delay_ms(1000);
```

```
PORTB=0x41; //letter A
```

```
PORTC |=(1<<PORTC)
```

```
PORTC &=~(1<<PORTC)
```

```
_delay_ms(1000);
```



شكل ٥-٥ تشغيل شاشة عرض LCD من خلال المتحكم atmega328

```
PORTB=0x44; //letter D
```

```
PORTC |=(1<<PORTC2);
```

```
PORTC &=~(1<<PORTC2);
```

```
_delay_ms(1000);
```

```
PORTB=0x41; //letter A
```

```
PORTC |=(1<<PORTC2);
```

```
POR TC &= ~(1<<PORTC2);
_delay_ms(1000);
```

```
POR TB=0x57; //letter W
POR TC |=(1<<PORTC2);
POR TC &= ~(1<<PORTC2);
_delay_ms(1000);
```

```
POR TB=0x59; //letter Y
POR TC |=(1<<PORTC2);
POR TC &= ~(1<<PORTC2);
_delay_ms(1000);
```

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

يمكنك الآن اللعب مع الشاشة عن طريق تغييرات بسيطة في البرنامج السابق مثل تأثير زمن التأخير بعد كل عملية تعامل مع الشاشة، والتحكم في إطفاء وإنارة دليل الكتابة، كما يمكنك تجربة نظام مسار البيانات ٤ بت بدلاً من ٨ بت. لقد رأينا من البرنامج السابق كيف أنه طويل جداً، وهل كلما احتجت لكتبة حرف أو حرفين على الشاشة وفي أي موضع فيها ساحتاج لكل هذا الكم من الأوامر؟ حيث كما تلاحظ فإنه لتنفيذ أي عملية على الشاشة فإنها تحتاج لأربع أوامر على الأقل. فمثلاً لكتابة أي حرف فإننا نضع كود الحرف، ثم ننشط خط التنشيط enable، ثم نحمد خط التنشيط مرة أخرى، ثم نقوم بعمل تأخير مناسب. فهل هناك وسيلة لتبسيط ذلك. يمكن تبسيط التعامل مع الشاشات بدرجة كبيرة عن طريق وضع البرنامج السابق في صورة مكتبة library تحتوى كل عمليات التعامل مع أي شاشة وسوف نسميها مثلاً LCDlib، ثم نقوم بتضمين هذه المكتبة في البرنامج الأساسي باستخدام أمر التضمين التالي: #include "LCDlib.h" كما سنرى في الجزء التالي.

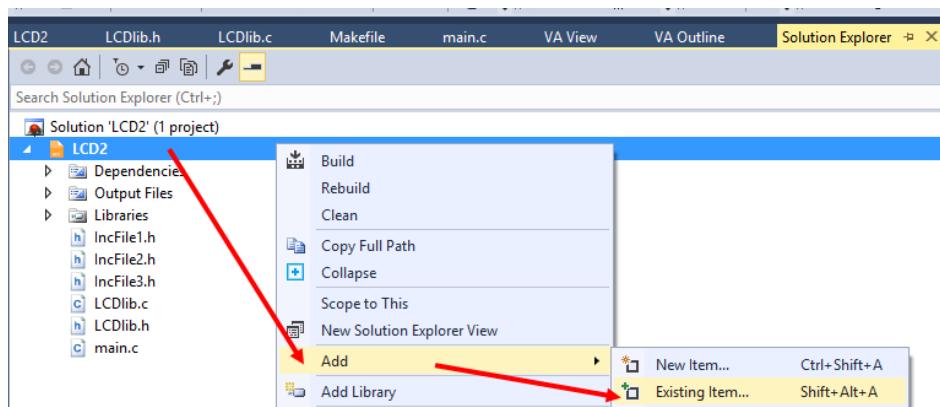
## ٣-٥ إنشاء مكتبة للتعامل مع الشاشات LCD

برنامج الأتميل استديو لا يحتوى مكتبة خاصة بالتعامل مع الشاشات LCD ولا مع أي ملحق من الملحقات التي يمكن توصيلها على المتحكمات بصفة عامة مثل الشاشات التي نحن بصددها هنا، أو التراسل التتابعى serial

communication، أو التعامل مع المحوّلات التماثلية الرقمية ADC، وهكذا الكثير من مثل هذه الملاحقات، ونحن سنشرح كيفية إنشاء مكتبة لكل واحد من الملاحقات التي ستعامل معها في هذا المقرر في الفصل الخاص به. هذه المكتبات يمكنك إعدادها بنفسك وإضافتها عندك لاستخدامها وقتما تحتاجها، أو تقوم بالبحث على الإنترنت عن مثل هذه المكتبات ثم تقوم بإضافتها للتطبيق الخاص بك، ونحن نفضل بالطبع الطريقة الأولى، لأنك في هذه الحالة ستكون المهنيمن على كل التطبيق الخاص بك وستكون على دراية بكل كبيرة وصغيرة عن هذا التطبيق. هذا بالإضافة إلى أنه عند تنزيلك لأحد المكتبات الجاهزة فإنه عليك أن تقرأها وتفهمها لكي تجعلها متوافقة مع التطبيق الخاص بك وطريقة التوصيل التي تتبعها، وهذا ربما يحتاج لوقت أطول مما لو قمت أنت بكتابتها من جديد.

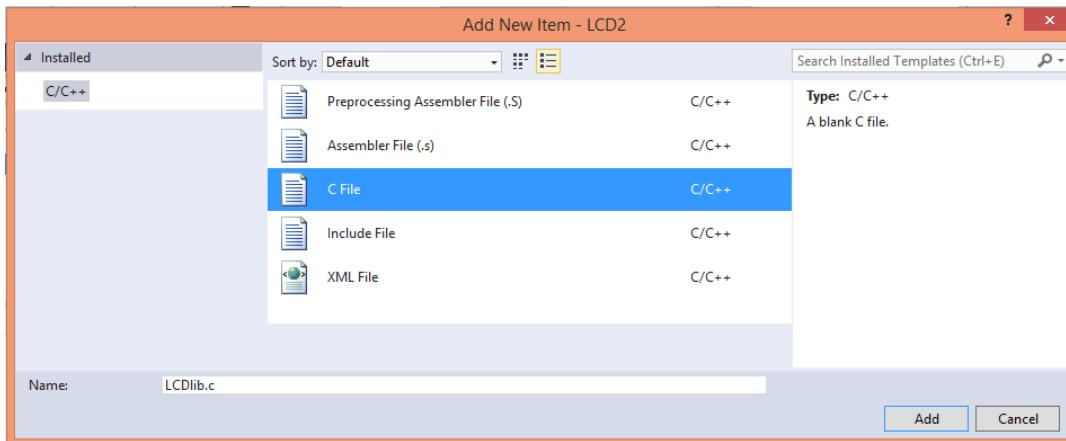
١- إذا كنت ستقوم بتنزيل هذه المكتبة من أي مصدر ولتكن الإنترنت مثلاً فإنه في هذه الحالة لابد من تنزيل ملفين يحملان نفس الإسم، أحدهما له الامتداد .c. ول يكن مثلاً LibLCD.c، والثاني سيكون LibLCD.h. بعد تنزيل هذين الملفين في أي مجلد على الحاسوب الخاص بك، إذهب للخطوة ٢.

٢- الآن من برنامج الأتمل استديو إضغط على الوصلة Solution Explorer وهذه الوصلة ستتجهنا في أي مكان على الأتمل استديو اعتماداً على طريقة تثبيتك للبرنامج وطريقة توزيعك لنوافذ البرنامج على شاشة الحاسوب. المهم بعد النقر على هذه الوصلة ستتجه وصلة أخرى بإسم المشروع الذي تعامل معه وأمام هذا الإسم أيقونة باللون البرتقالي كما في شكل ٦-٥. في شكل ٦-٥، إسم التطبيق الذي نتعامل معه هو LCD2. أنقر نقرة واحدة على إسم التطبيق لاختيارة فقط، ثم إضغط على الزر الأمن للماوس حيث ستسقط قائمة أحد مكوناتها هو الاختيار Add، وبالوقوف عليها ستسقط لك قائمة خيارات أخرى، أهم مكوناتها هو New Item و Existing Item ستحتار منها الثنائي وهو Existing Item حيث ستفتح لك نافذة يمكنك منها تبع المسار إلى أن تصل إلى مكان الملفين .c و .h. أنقر عليهما لإضافتهما.



شكل ٦-٥ إضافة ملف مكتبة موجود أصلاً في أحد المجلدات على الحاسوب

٣- الطريقة الثانية وهى بافتراض أنك ستقوم بإعداد هذه الملفات بنفسك. في هذه الحالة ستبعد الخطوة ٢ السابقة كما في شكل ٦-٥ ولكن في هذه الحالة سننقر على الاختيار New Item باعتبار أننا سنكتب هذين البرنامجين بأنفسنا. في هذه الحالة ستظهر لك الشاشة الموضحة في شكل ٧-٥.



شكل ٧-٥ فتح ملف مكتبة جديد بالامتداد .c.

٤- في خانة إسم الملف في أسفل النافذة الموضحة في شكل ٧-٥ أكتب إسم ملف المكتبة، وسنفترضه LCDlib.c ثم انقر على الزرار Add حيث سيتم فتح ملف جديد بهذا الإسم في مساحة تحرير الأقليل استديو سنقوم بكتابة برنامج مكتبة الشاشة في هذه المساحة كما يلى:

```
/*
 * LCDlib.c
 *
 * Created: 6/22/2017 4:55:36 PM
 * Author: Mohamed Eladawy
 */
#ifndef F_CPU
#define F_CPU 1000000ul
#endif
#include <avr/io.h>
#include <util/delay.h>
#include "lcdlib.h"
```

سنفترض هنا أن مسار بيانات الشاشة سيوصى على البوابة B //  
وأن خطوط التحكم الثلاثة RS و WR و EN سيتم توصيلها على أول ثلاث خطوط من البوابة  
// C

مسجل الاتجاه لتحديد اتجاه بوابة مسار بيانات الشاشة//  
مسجل الاتجاه لتحديد اتجاه مسار التحكم للشاشة//  
تحديد البوابة التي سيتم مسار البيانات للشاشة عليها//  
تحديد البوابة التي سيتم توصيل خطوط التحكم عليها//  
رقم البت التي سيوصى عليها خط التحكم RS //  
رقم البت التي سيوصى عليها خط التحكم RW //  
رقم البت التي سيوصى عليها خط التحكم EN //

/\* بفرض أن الشاشة المستخدمة تحتوى على سطرين فقط كل منهما 16 حرفا \*/

```
#define LCDMaxLines 2
#define LCDMaxChars 16
#define LineOne 0x80
#define LineTwo 0xc0
#define BlankSpace ''
```

/\* هذه الدالة خاصة بتجهيز الشاشة لعمل بطريقة 8 بت \*/

```
void LCD_Init()
{
    _delay_ms(50);
```

تحديد البوابة B والبوابة C ليكونا بوابات خرج//

```
dataBus_direction = 0xff;
controlBus_direction = 0xff;
```

نداء على هذه الدالة بهذه الشفرة للتعرف بشاشة من سطرين وكل حرف //

مصفوفة 7×5

LCD\_CmdWrite(0x0E);      نداء على هذه الدالة بهذه الشفرة لتشغيل الشاشة وإظهار دليل //  
الكتاب

LCD\_CmdWrite(0x01);      مسح الشاشة من أى حروف //

وضع الدليل عند أول مكان في أول سطر //  
LCD\_CmdWrite(0x80);  
}

هذه الدالة ترسل أمر إلى الشاشة //  
void LCD\_CmdWrite( char cmd)

وضع الأمر على مسار البيانات //  
databus=cmd;  
التسجيل في مسجل الأوامر //  
control\_bus &=~(1<<rs);  
تنشيط عملية الكتابة //  
control\_bus &=~(1<<rw);  
جعل خط التنشيط يساوى واحد //  
control\_bus |=1<<en;  
\_delay\_ms(1);  
جعل خط التنشيط يساوى صفر //  
control\_bus &=~(1<<en);  
\_delay\_ms(1);

{

هذه الدالة ترسل حرف للشاشة لإظهاره//  
void LCD\_DataWrite( char dat)

{

وضع الحرف على مسار البيانات //  
databus=dat;  
التسجيل في مسجل البيانات //  
control\_bus |=1<<rs;  
تنشيط عملية الكتابة //  
control\_bus &=~(1<<rw);  
جعل خط التنشيط يساوى واحد //  
control\_bus |=1<<en;  
\_delay\_ms(1);  
جعل خط التنشيط يساوى صفر //  
control\_bus &=~(1<<en);  
\_delay\_ms(1);

{

في هذا البرنامج الأول من الأسطر ١٤ يضعها الأتمل استديو تلقائيا، ويمكنك التعديل فيها بما يناسبك إن أردت، ومن ذلك مثلاً تردد وحدة المعالجة المركزية الذي يتم وضعه يساوى 1000000 نبضة في الثانية. يأتي بعد ذلك في السطرين ١٥ و ١٦ تضمين بعض المكتبات القياسية بالأمرین: #include <util/delay.h> و #include <avr/io.h> و نقصد بالمكتبات القياسية أنها مكتبات موجودة ضمن الأتمل استديو وليس من عمل المستخدمين، ولقد رأينا مسبقاً لماذا نقوم بتضمين كل واحد من هذين الملفين،

فال الأول ضروري لأنه يحتوى التعريف بأسماء بوابات المتحكم، والثانى ضروري لأنه يحتوى المكتبة المستخدمة فى عمل أزمنة التأخير. يأتي بعد ذلك أمر تضمين المكتبة التى نحن بصددها (مكتبة الشاشة) وهو الأمر ١٧ كالالتى : #include "lcdlib.h". لاحظ الفرق بين تضمين المكتبة القياسية التى يتم وضعها بين القوسين < ، والمكتبة المعرفة من قبل المستخدم والتى يتم وضع إسمها بين علامتى تنصيص " " .

بعد ذلك تم التعريف ببعض الثوابت من خلال الأمر #define وقد قمنا بكتابة وظيفة كل أمر بالعربي بجوار كتعليق لهذا الأمر حتى لا نعيد شرحها مرة ثانية. بعد ذلك تم تحديد ثلاث دوال الدوال () LCD\_Init() و LCD\_DataWrite( char dat ) و LCD\_CmdWrite( char cmd ) من حيث عدد الأسطر فيها والمصفوفة التى يكتب فيها كل حرف، كما هو موضح في التعليقات المكتوبة بجوار كل أمر فيها. الدالة الثانية تقوم بإرسال أمر إلى مسجل الأوامر للشاشة والتعليقات أيضا مكتوبة بجوار كل سطر، وأما الدالة الثالثة فتقوم بإرسال حرف إلى مسجل البيانات لإظهاره على الشاشة والتعليقات بجوار كل أمر توضح ذلك أيضا. سنكتفى هنا بهذه الدوال الثلاثة وهناك الكثير من الدوال الأخرى الموضحة في البرنامج LCDlib.c في الملحق الأول في آخر الكتاب والتي توضح الكثير من العمليات التي يمكن إجراؤها على الشاشة LCD. بهذا تكون قد انتهينا من كتابة الملف الأول من ملفي المكتبة LCD وهو الملف LCDlib.c.

٥ - بعد ذلك نبدأ في إضافة الملف الثاني LCDlib.h بنفس طريقة إضافة الملف الأول LCDlib.c كما في شكل

٧-٥ ولكن هذه المرة ننقر بالماوس على الاختيار include file بعد تسمية الملف بنفس الإسم والنقر على الاختيار Add حيث ستظهر شاشة تحرير جديدة باسم هذا الملف تقوم بكتابة الأوامر التالية فيها:

```
/*
* LCDlib.h
*
* Created: 6/24/2017 6:56:37 AM
* Author: Mohamed Eladawy
*/
#ifndef LCDLIB_H_
#define LCDLIB_H_
#define databus_direction DDRB
#define controlbus_direction DDRC
#define databus PORTB
#define control_bus PORTC
```

```

#define rs 0
#define rw 1
#define en 2
/* 16x2 LCD Specification */
#define LCDMaxLines 2
#define LCDMaxChars 16
#define LineOne 0x80
#define LineTwo 0xc0

#define BlankSpace ''
void LCD_Init();
void LCD_Clear();
void LCD_GoToLineOne();
void LCD_GoToLineTwo();
void LCD_GoToXY(char row, char col);
void LCD_CmdWrite( char cmd);
void LCD_DataWrite( char dat);
void LCD_DisplayString(char *string_ptr);
#endif /* LCDLIB_H_ */

```

لاحظ أن الملف LCDlib.h عبارة عن تعريفات بكل الثوابت والدوال المستخدمة في الملف LCDlib.c والتي تم التعريف بها في تعليقات هذا الملف. بذلك تكون قد انتهينا من إضافة ملفي المكتبة الخاصة بالشاشات LCD ويمكن الآن استخدامها في البرنامج الأساسي LCD2 كما يلى لكتابة الرسالة التالية: "Hello World" بحيث يتم كتابة الكلمة Hello حرف بحرف في السطر الأول، ثم نقوم بكتابة الكلمة الثانية World كسلسلة أحرف مرة واحدة في السطر الثاني. سنستخدم نفس الدائرة الموضحة في شكل ٥-٥ لإظهار هذه الرسالة الجديدة بهذه الطريقة الجديدة باستخدام المكتبة LCDlib التي تم إضافتها. البرنامج الأساسي الذي سيقوم بهذه المهمة سيكون كالتالى:

```

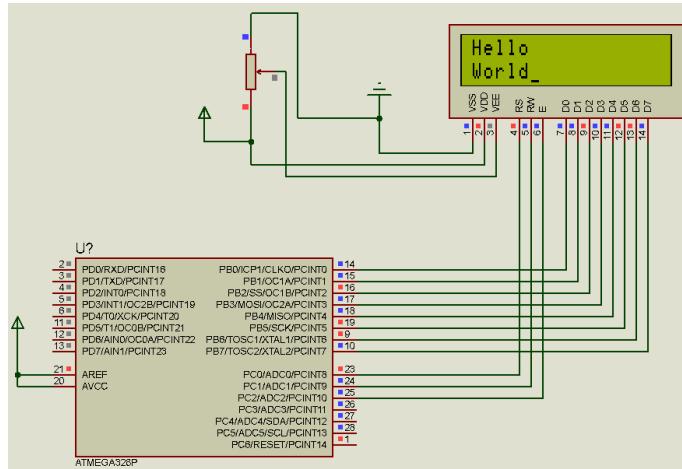
/*
* LCD2.c
*
* Created: 6/20/2017 2:20:03 PM
* Author : Mohamed Eladawy

```

```
*/
#define F_CPU 1000000ul
#include <avr/io.h>
#include <util/delay.h>
#include "LCDlib.h"

int main(void)
{
    LCD_Init();
    LCD_DataWrite('H');
    LCD_DataWrite('e');
    LCD_DataWrite('l');
    LCD_DataWrite('l');
    LCD_DataWrite('o');

    LCD_GoToLineTwo();
    LCD_DisplayString("World");
    while (1)
    {
    }
}
```



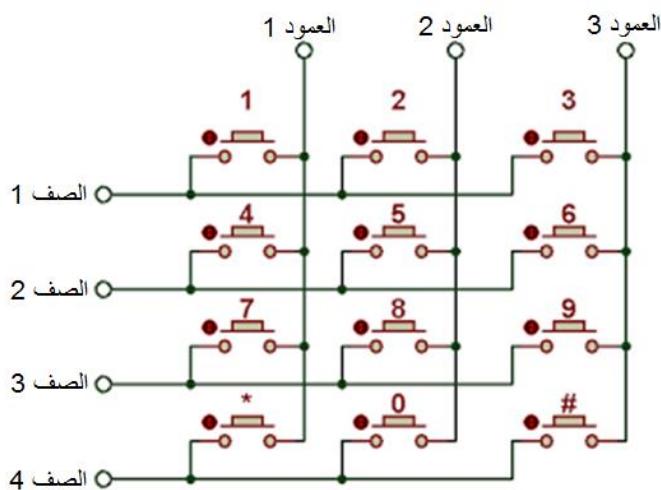
شكل ٨-٥ العرض على الشاشة LCD باستخدام المكتبة LCDlib التي تم إضافتها

لاحظ كيف أن برنامج التعامل مع الشاشة تقلص بدرجة كبيرة مع استخدام ملف المكتبة LCDlib الذي قمنا بإضافته، وقمنا بتوضيح نتيجة استخدامه كما في شكل ٨-٥. تذكر أن المكتبة LCDlib أصبحت الآن متوافرة لتضمينها مع أي برنامج آخر مجرد كتابة الأمر `#include "LCDlib.h"` كما سبق ودون الحاجة لإعادة كتابتها من جديد. لاحظ أيضاً أنها لعرض الكلمة الأولى "Hello" استخدمنا الدالة `LCD_DataWrite('e')` التي تعرض حرفاً واحداً، بينما لعرض الكلمة الثانية استخدمنا الدالة `LCD_DisplayString("World")` التي تعرض سلسلة كاملة من الأحرف، وهذه الدالة ودوايا أخرى خاصة بالتعامل مع الشاشة LCD موضحة في الملحق الأول.

## ٥-٤ التعامل مع مصفوفة المفاتيح

مصفوفة المفاتيح عبارة عن مجموعة من المفاتيح الضاغطة push buttons يتم وضعها في صورة صفوف وأعمدة كما في شكل ٩-٥ الذي يبين مصفوفة مكونة ١٢ مفتاحاً مرتبة في أربع صفوف وثلاث أعمدة. في العادة يتم توصيل الأعمدة على الجهد  $V_{CC}$  من خلال مقاومة جذب مقدارها ٥ كيلوأوم، وأما الصفوف فيتم توصيلها بالأرضي. عند ضغط أي مفتاح يتم توصيل أحد الصفوف بأحد الأعمدة التي يشترك معها هذا المفتاح بحيث يمر التيار من مصدر الجهد إلى المقاومة ثم المفتاح ثم الأرضي، ولذلك فعند قراءة العمود من نقطة تلامس المفتاح المضغوط تعطى صفراء، بينما قراءة أي عمود لم يتم ضغط أي مفتاح عنده فإنه يعطي واحداً (جهد عالي).

لكى نستخدم المتحكم لمعرفة أي المفاتيح يكون مضغوطاً من بين هذه المفاتيح فإننا نقوم بتوصيل الصفوف على بوابة خرج، ولتكن مثلاً البوابة D، (بالطبع سنستخدم الأربع خطوط الأولى منها فقط). أما الأعمدة فسنقوم بتوصيلها على



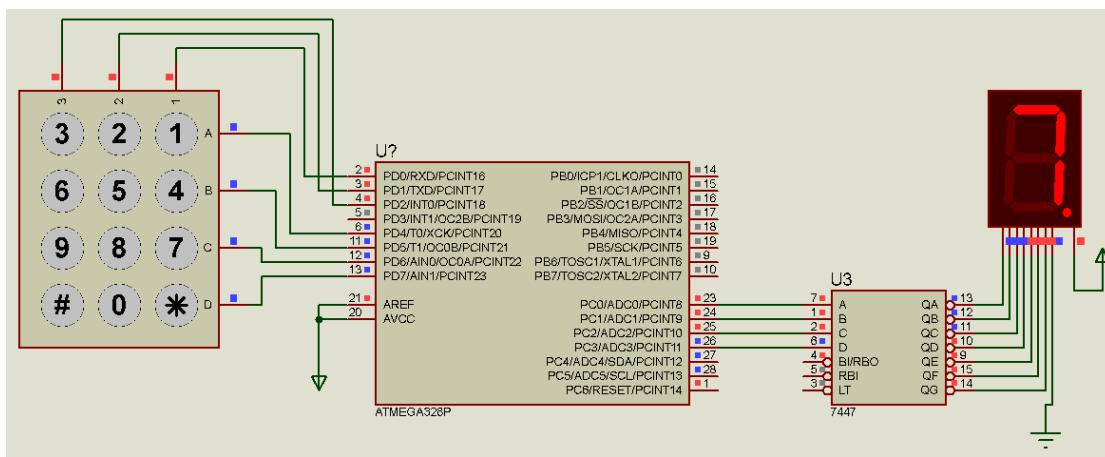
شكل ٩-٥ مصفوفة مفاتيح من ٤ صفوف و ٣ أعمدة

بوابة إدخال، ولتكن مثلاً البوابة C، (وبالطبع سنستخدم الثلاث خطوط الأولى منها فقط، أو حتى النصف الثاني من البوابة D). البرنامج الذى سنكتبه يقوم بوضع صفر على الصف ١، ثم يقرأ الأعمدة، فإن وجد أن أحد الأعمدة يساوى صفر فهذا يعني أن المفتاح الذى بين هذا الصف وهذا العمود مضغوط. فمثلاً إذا وجد أن العمود ٢ يساوى صفر، فهذا يعني أن المفتاح ٢ هو المضغوط، حيث في هذه الحالة يقوم البرنامج بعمل الفعل المراد عمله في هذه الحالة ولتكن مثلاً إظهار الرقم ٢ على مظهر ذو ٧ قطع أو شاشة

LCD. بعد ذلك ينتقل البرنامج إلى الصف الثاني ليضعه بصفر ثم يقرأ الأعمدة ليحدد هل المفتاح ٤ أم ٥ أم ٦ هو المضغوط. بنفس الطريقة ينتقل البرنامج إلى الصف الثالث، ثم الرابع، ثم يعيد الكرة من جديد ليقوم بتصفيير الصفوف بدأ من الصف الأول حتى الرابع، وهكذا إلى مالا نهاية.

في هذا المشروع سنقوم بتوصيل مصفوفة المفاتيح على البوابة D بحيث يتم توصيل الصفوف الأربع على الأربع باتات الأخيرة من هذه البوابة، وبالتالي فإن هذا النصف من البوابة D سيتم تحديده كخطوط خرج. في نفس الوقت نقوم

بتوصيل الأعمدة الثلاثة على أول ثلاث باتات منها كما في شكل ١٠-٥، وسنحدد هذه الخطوط من البوابة D كخطوط دخل. لكي نختبر المفاتيح التي يتم ضغطها سنوصل شريحة مظهر ذو ٧ قطع بحيث يظهر عليه رقم المفتاح المضغوط، وهذا المظهر سيتم توصيله من خلال شريحة دافع تيار محمول شفرات من الصورة الثانية المكودة عشريا إلى شفرات السبع قطع وهو الشريحة 7447 ( يمكنك مراجعة عمل هذه الشريحة في كتاب الدوائر المنطقية للمؤلف والمتاح على الإنترنت) كما في شكل ١٠-٥ . الشفرات الثنائية المكودة عشريا والتي تمثل المفتاح المضغوط سيتم أخذها من النصف الأول من البوابة C التي سيتم تحديدها كبوابة إخراج.



شكل ١٠-٥ إظهار المفاتيح المضغوطة من مصفوفة المفاتيح على مظهر ذو ٧ قطع

```
/*
* KeyPad1.c
*
* Created: 6/25/2017 10:05:07 PM
* Author : Mohamed Eladawy
*/
#include <avr/io.h>
#define F_CPU 1000000UL
#include <util/delay.h>
#define ROW PORTD
#define COL PIND
unsigned char key;
unsigned char ScanKey;
```

تحديد النصف الثاني من D كخطوط خرج وتعريفها بالاسم ROW، والنصف الأول من D كخطوط دخل وتعريفها بالاسم .COL.

```
unsigned char i;
```

```
int main(void)
{
    DDRD=0xf0;
    DDRC=0xff;
    while (1)
    {
        do
        {
            ROW=0x0f;
            key=COL & 0x07;
        }while(key!=0x07);
        do
        {
            do
            {
                ROW=0x07;
                key=COL & 0x07;
            }while(key==0x07);

            _delay_ms(1);

            ROW=0x07;
            key=COL & 0x07;

        }while(key==0x07);
        ScanKey = 0xe0;
        for(i=0;i<0x04;i++)
        {
            ROW=ScanKey + 0x07;
            key=COL & 0x07;
            if(key!= 0x07)
                break;
            ScanKey=(ScanKey<<1)+ 0x10;
        }
    }
}
```

حلقة do..while تضع أصفار على كل الصفوف ووحيد على الأعمدة ، ثم قراءة الأعمدة الثلاثة بالأمر key=COL&0x07 وطالما أن أي مفتاح مضغوط لا يعمل شيء حتى يتم الانتهاء من ضغط المفتاح. الأعمدة الثلاثة ستتساوى 0x07 طالما لم يتم ضغط أي مفتاح.

حلقة do..while متداخلتين للتخلص من اهتزازات أي مفتاح عند ضغطه.

حلقة على الأربع صفوف لمعرفة أي مفتاح تم ضغطه، وتحديد الصنف والنصف الأول من الأعمدة، بتنصفها الثاني هو

```
شفرة المفتاح المضغوط من ٨ بت // Bit 8
```

```
switch(key) // Decode the key
```

```
{
```

```
case 0xe6: PORTC=0x01; break;
case 0xe5: PORTC=0x02; break;
case 0xe3: PORTC=0x03; break;
case 0xd6: PORTC=0x04; break;
case 0xd5: PORTC=0x05; break;
case 0xd3: PORTC=0x06; break;
case 0xb6: PORTC=0x07; break;
case 0xb5: PORTC=0x08; break;
case 0xb3: PORTC=0x09; break;
case 0x76: PORTC=0x0A; break;
case 0x75: PORTC=0x00; break;
case 0x73: PORTC=0x0b; break;
default: PORTC=0x00;
```

```
}
```

```
}
```

بالنسبة لجزء التخلص من الاهتزازات التي تنشأ عند الضغط على أي مفتاح ميكانيكي فإن الحلقة الأولى do..while تضع أصفارا على كل الصفوف ووحيد على الأعمدة بالرقم 0x07، ثم قراءة الصفوف بالأمر key=COL & 0x07؛ وطالما أن قيمة المفتاح تكون هي القيمة 0x07 فإن ذلك يعني أنه ليس هناك مفتاح مضغوط فلا يعمل شيء ويظل يدور في هذه الحلقة. إذا تم ضغط أي مفتاح، فإن قيمة المفتاح ستختلف عن القيمة 0x07، وبالتالي يخرج البرنامج من هذه الحلقة. بعد الخروج من الحلقة يتم عمل تأخير مقداره واحد ميلليثانوي، ثم وضع الرقم 0x07 مرة ثانية على الصفوف والأعمدة وقراءة الأعمدة مرة ثانية بنفس الأمر key=COL & 0x07؛ فإذا رجعت قراءة المفتاح key إلى القيمة 0x07 مرة ثانية، فإن ذلك يدل على أن الذي حدث كان مجرد هزة مفتاح وبناء عليه يعود البرنامج للدوران في الحلقة الأولى، وأما إذا ظلت قيمة المفتاح key مختلفة عن القيمة 0x07 فإن ذلك يعني أنه تم بالفعل ضغط أحد المفاتيح، وعليه فإن البرنامج سيخرج من الحلقة وينتقل إلى مرحلة تحديد المفتاح المضغوط ثم مرحلة فك شفرته. عندما يصل البرنامج لهذه المرحلة فإن ذلك يعني أن هناك مفتاح قد تم ضغطه، ولكننا لا نعرف أي واحد من المفاتيح الائنة عشرة قد تم ضغطه. تعالى نفترض أنها قمنا بضغط المفتاح 5 الموجود عند تقاطع الصفيحة الثانية والعمود الثاني. في

فك شفرة المفتاح المضغوط وإخراج الأربع بنايات المقابلة على البوابات لإظهارها

البداية يتم وضع المتغير ScanKey = 0xe0=11100000، حيث 0xe0=11100000 مما يعني وضع الصف الأول بالقيمة صفر (باللون الأحمر). ثم الدخول في حلقة ( ) for من أربع مرات. في الأمر الأول في هذه الحلقة يتم تنفيذ الأمر التالي:  $ROW=ScanKey + 0x07;$  حيث ستكون القيمة ROW التي سيتم إخراجها على البوابة D هي key=COL & 0x07=11100000+00000111=11100111، وبعد ذلك يتم تنفيذ الأمر التالي مباشرة: key=COL=00000111، وحيث أن الصف الأول فقط هو الفعال (يساوي صفر)، فإن قراءة الأعمدة في هذه الحالة ستكون key=COL=00000111، أي 0x07، وبالتالي فإن نتيجة عملية الآند ستكون هي نفس القيمة 0x07 وستكون قيمة المفتاح key=0x07، وبالتالي ستكون نتيجة تنفيذ الأمر الشرطى if غير محققة، وسينتقل التنفيذ إلى الأمر التالي وهو:  $ScanKey=(ScanKey<>1)+0x10$ .

هذا الأمر سيزيد القيمة ScanKey ناحية اليسار بمقدار بت واحدة، وبالتالي فإن نتيجة تنفيذ هذا الأمر ستصبح كالتالي: 11010000+00010000=11010000، أي أن الصف الثاني أصبح الآن يساوى صفر (الأحمر). بعد ذلك يرجع التنفيذ لبداية الحلقة for حيث ينفذ الأمر  $ROW=ScanKey + 0x07$  الذي سيعطي  $ROW=11010000+00000111=11010111$  حيث القيمة key=COL & 0x07=11010111. بعد ذلك يتم تنفيذ الأمر key=COL=00000101 نتيجة ضغط الزرار 5 الذى في العمود الثانى وفي الصف الثانى الذى يساوى صفر من الأمر السابق. وعلى ذلك ستكون قيمة المفتاح key=00000101 أي لا تساوى 0x07، وعلى ذلك ستكون نتيجة تنفيذ أمر الشرط if التالى حقيقية، وبالتالي يتم تنفيذ الأمر break; الذى يتبع عنه الخروج من الحلقة ( ) for وينفذ أول أمر بعد الحلقة وهو;  $key = key + (ScanKey & 0xf7)$  الذى ينتج عنه أن قيمة المفتاح ستصبح:

$Key=00000101+(11010000&11110111)=11010101=0xd5$  وهذه هي شفرة المفتاح التى تحتوى الصف والعمود الواقع فيه هذا المفتاح. بعد ذلك يدخل البرنامج فى فك شفرة المفتاح للتعرف عليه من خلال الأمر  $\{ switch(key) \}$  حيث سيذهب إلى الحالة d5 التى ستجعل البوابة PORTC=05 التى ستظهر على المظهر ذو السبع قطع كما فى البرنامج. بنفس الطريقة يمكن تتبع أي ضغطة لأى مفتاح. حاول كتابة البرنامج وتنفيذته والتحقق من صحته.

يمكن إعادة كتابة البرنامج السابق فى صورة دوال يمكن النداء عليها من خلال ملف سنسيميه Keypadlib.h تقوم بتضمينه فى البرنامج الأساسى بعد أن نضيف ملف جديد سنسيميه أيضا Keypadlib.c عن طريق إضافتها من خلال مستكشف الحل Solution Explorer كما حدث فى الملفين LCDlib.h و LCDlib.c الذين تم شرحهما من قبل. البرنامج الأساسى بعد إعادة كتابته فى صورة دوال يتم النداء عليها من الملف المكتوب Keypadlib.h سيكون كما يلى:  
/\*

```
* KeyPad2.c
*
* Created: 6/27/2017 12:12:08 PM
* Author : Mohamed Eladawy
*/
```

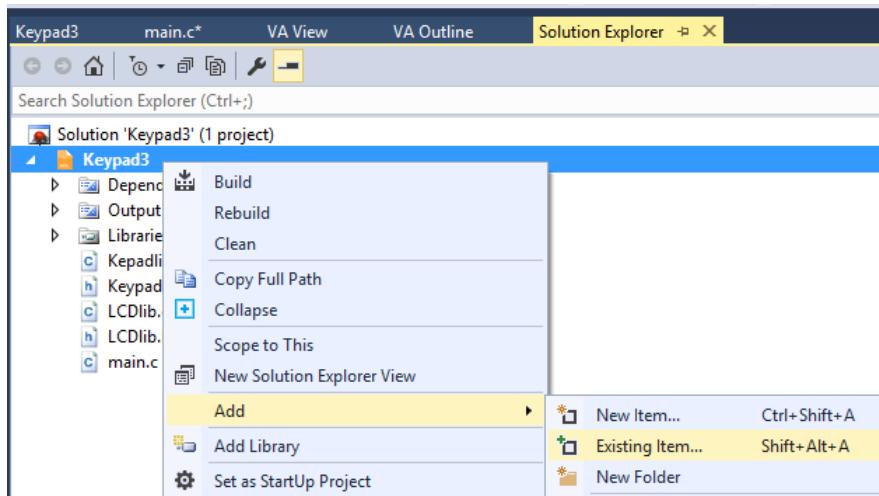
```
#include <avr/io.h>
#define F_CPU 1000000ul
#include <util/delay.h>
#include "Keypadlib.h"
#define ROW PORTD
#define COL PIND

int main(void)
{
    KEYPAD_Init();
    WaitForKeyRelease();
    WaitForKeyPress();
    while (1)
    {
        GetKey();
    }
}
```

الملفان Keypad.c و Keypadlib.h موجودان بالكامل في الملحق ٢ في آخر الكتاب يمكنك نسخهما والاستفادة منهما.

الآن سنقوم بإظهار المفاتيح التي يتم ضربها على مصفوفة المفاتيح على شاشة LCD. بالطبع سنستفيد هنا من الملف المكتبي LCDlib.h الذي أعددناه في المشروع Lcd2 السابق عن طريق تضمينه في المشروع الجديد. كذلك سنستفيد من الملف المكتبي الخاص بمصفوفة المفاتيح Keypadlib.h الذي أعددناه في المثال السابق Keypad2.c. بعد أن نفتح المشروع الجديد Keypad3.c نقوم بإضافة الملفات Keypadlib.c و LCDlib.h و Keypadlib.h و LCDlib.c عن keypadlib.h.

نقر على مستكشف الحل Solution Explorer واتبع نفس الخطوات الموضحة في شكل ١١-٥ .



شكل ١١-٥ إضافة الملفات المكتبية الموجودة أصلاً في مشروع آخر

يوضح شكل ١١-٥ أننا نتبع نفس الخطوات المتبعة في إضافة ملف جديد ولكننا نختار هذه المرة Existing Item حيث أن هذه الملفات موجودة أصلاً وسبق تنفيذها. لذلك بمجرد النقر على الاختيار Existing Item سيفتح لك الأتميل استديو قائمة المشاريع السابقة، تقوم أنت بالنقر على المشروع الذي يحتوى هذه الملفات المكتبية إلى أن تصل إلى كل منها فتنقر عليه مرتين حيث تم إضافته إلى مشروعك الجديد. البرنامج الجديد سيكون كالتالي:

```
/*
 * Keypad3.c
 *
 * Created: 6/27/2017 2:45:26 PM
 * Author : Mohamed Eladawy
 */
#include <avr/io.h>
#define F_CPU 1000000ul
#include <util/delay.h>
#include "LCDlib.h"
#include "Keypadlib.h"
#define ROW PORTD
#define COL PIND
unsigned char key;
int main(void)
{
```

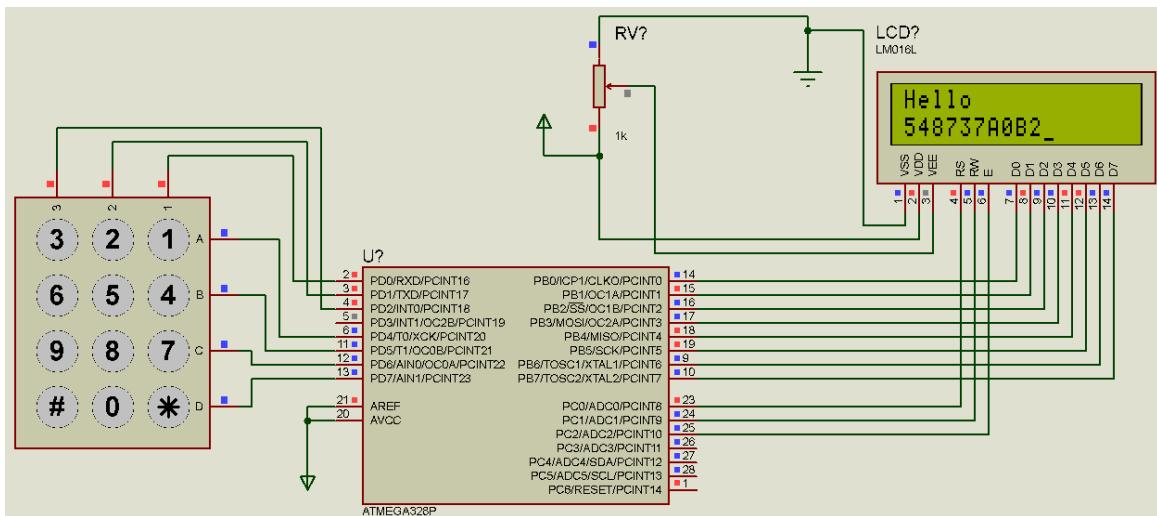
```

KEYPAD_Init();
LCD_Init();
LCD_DataWrite('H');
LCD_DataWrite('e');
LCD_DataWrite('l');
LCD_DataWrite('l');
LCD_DataWrite('o');
LCD_GoToLineTwo();
WaitForKeyRelease();
WaitForKeyPress();

while (1)
{
    key=GetKeyForLCD();
    LCD_DataWrite(key);
}

```

كما نرى فإن هذا البرنامج ينادى على دالى تجهيز مصفوفة المفاتيح والشاشة LCD وهما الأمران `KEYPAD_Init()` و `LCD_Init()` ثم بعد ذلك يتم كتابة الكلمة Hello في السطر الأول من الشاشة حرف بحرف بالأمر `LCD_GoToLineTwo()`; بعد ذلك جعلنا دليل الكتابة ينتقل للسطر الثاني بالأمر `LCD_DataWrite('H');` وهو أحد دوال الملف المكتبي LCDlib.c حيث يمكنك مراجعته في الملحق الأول. بعد انتقال دليل الكتابة cursor إلى السطر الثاني سينتظر البرنامج لضرب أي مفتاح حيث يدخل في الحلقة while(1) وفيها يقرأ أي مفتاح يتم ضغطه بالأمر `key=GetKeyForLCD();` ثم يظهره على الشاشة LCD، في السطر الثاني من الشاشة. الجديد هنا أن الدالة `GetKeyForLCD()` تختلف عن مثيلتها في الملف المكتبي السابق وهي `GetKey()` حيث أن هذه الدالة تحصل على شفرة المفتاح في الصورة الثنائية المكونة عشريا BCD وترسلها إلى البوابة C لعرضها على المظهر ذو السبع قطع. أما هنا فلكل مفتاح على الشاشة LCD لابد أن تكون شفرته هي شفرة الأسكنى. لذلك قمنا بإضافة دالة جديدة تقرأ شفرة المفتاح وتضعها في الصورة الأسكنى وتعود بهذه القيمة الجديدة، وهذه الدالة الجديدة أسميناها `GetKeyForLCD()` ويمكنك مراجعتها في الملف المكتبي Keypadlib.c في الملحق الثاني. شكل ١٢-٥ يبين محاكاة عمل مصفوفة المفاتيح مع الشاشة LCD والتي تعمل بالبرنامج السابق Kepad3.c.



شكل ١٢-٥ توصيل مصفوفة المفاتيح على الشاشة LCD

نكتفى بهذا القدر من الحديث عن التعامل مع الشاشات LCD والمظاهر ذات السبع قطع ومصفوفة المفاتيح وترك للقارئ اللعب بهذه الأشياء بنفسه واستخدام أنواع أخرى من هذه الشاشات ومصفوفات المفاتيح وعليك فقط أن تعدل من الملفات المكتبية المستخدمة لتناسب الأنواع الجديدة التي ستستخدمها.

## ملخص الفصل

لقد تم التركيز في هذا الفصل على تطبيقين شهيرين وكثيري الاستخدام وهما توصيل شاشات البليور السائلة LCD على المتحكم، وتوصيل مصفوفة مفاتيح صغيرة، وشاشة عرض 7 قطع أيضا. من المواضيع المفيدة أيضا في هذا الفصل هو كيفية كتابة الملفات المكتبية بصورة عامة بحيث يمكن تضمينها مع أي برنامج آخر بحيث لا تكون مضطرين لكتابة هذه الملفات من جديد مع أي تطبيق يحتاجها. بهذه الطريقة يمكن أن يكون لديك مكتبة خاصة التي تحتوى الكثير من الملفات التي يمكن تضمينها مع الكثير من التطبيقات عند الحاجة إليها.

## الفصل ٦

### المقاطعة الخارجية

### External Interrupt

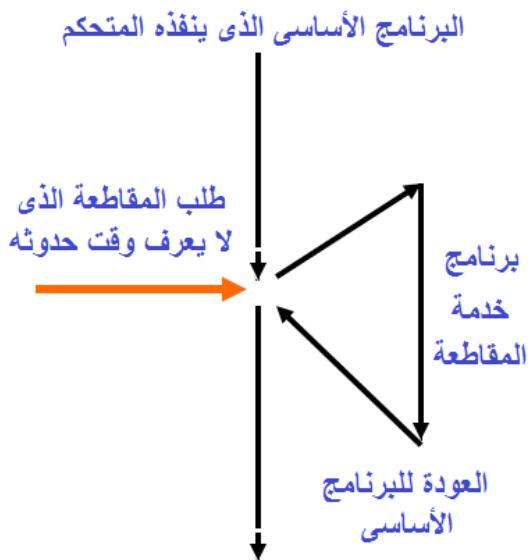
**العناوين المضيئة في هذا الفصل:**

- ١- متجه المقاطعة
- ٢- المقاطعة الخارجية من على الطرفين INT0 و INT1
- ٣- المقاطعة بسبب التغير على أي طرف
- ٤- شرح مشروع جراج سيارات

## ٦-١ مقدمة

سنلقي في هذا الفصل نظرة شاملة عن المقاطعة بصورة خاصة على المقاطعة الخارجية للمتحكم atmeg328 الذي نعتمد عليه كأداة برمجة في هذا الكتاب. نقصد بالمقاطعة الخارجية بأنها المقاطعة التي تعطى خارجيا

على أحد أطراف المتحكم. أما المقاطعة الداخلية التي تأتي من قبل الملحقات الداخلية في المتحكم مثل المؤقتات والمتحول التماضي الرقمي وغيرها فسيأتي الحديث عن كل منها في الفصل الخاص بذلك.



شكل ٦-١ رسم تخطيطي لمقاطعة المتحكم

معنى المقاطعة يمكننا تقديمها من الترجمة الحرافية له من موقع ويكيبيديا الشهير الذي يعرف المقاطعة بأنها عبارة عن إشارة إلى المعالج أو المتحكم آتية من مصدر عتادى hardware أو برمجي software توضح وجود حدث يستوجب الانتباه أو الخدمة الفورية من قبل المعالج أو المتحكم. في الحال يقوم المتحكم بتعليق نشاطه الحالى ويدهب لخدمة هذه المقاطعة، وبعد الانتهاء من هذه الخدمة يعود لاستئناف نشاطه من نفس

المكان الذى توقف عنده. كمثال على ذلك وأنت تتحدث مع زميل لك فى أحد المواضيع ثم يأتي أحدهم فجأة ليسألك عن مكان الحاضرة مثلا. في هذه الحالة ستتوقف عن الحديث مع زميلك ثم تجيب السائل، وبعد إجابة السائل تعود لاستئناف الحديث مع زميلك. شكل ٦-١ يبين رسمًا تخطيطيًا لمقاطعة المعالج أو المتحكم. تبرز هنا مجموعة من الأسئلة التي يجب الإجابة عليها لكي نفهم هذا الموضوع وسنحاول الإجابة على كل منها باختصار دون الدخول في التفاصيل الدقيقة.

### ما هي الأنواع المختلفة لمقاطعة؟

يقسم المتخصصون المقاطعة إلى أنواع عديدة وبطرق مختلفة. فالبعض مثلاً يقسم المقاطعة إلى مقاطعة عتادية hardware interrupt وأخرى برمجية software interrupt كما أشرنا مسبقاً. المقاطعة الأولى يكون مصدرها عتادياً وتحدث عادةً من خلال أطراف المتحكم أو المعالج. من أمثلة ذلك أن يتم مقاطعة المتحكم في حالة حدوث حريق مثلاً أو ارتفاع في درجة الحرارة حيث تتم هذه المقاطعة على أحد أطراف المتحكم، وعند قبول المقاطعة يذهب

المتحكم إلى برنامج خدمة المقاطعة الذي يقوم بفعل معين مناسب لحالة هذه المقاطعة مثل ضرب جرس إنذار أو استدعاء مطافىء أو غير ذلك. من أمثلة هذه الأطراف الموجودة في جميع المعالجات والمتحكمات تقريباً الطرف Reset والذى بتنشيطه يترك المتحكم أو المعالج ما ينفذه ويذهب لتنفيذ برنامج معين خاص بإعاده وضع المعالج أو المتحكم. أما المقاطعة البراجميه فهى تحدث لأسباب براجميه تحدث أثناء تنفيذ البرنامج مثل القسمة على الصفر مثلاً، التي قد تحدث عرضياً حيث عندها يخرج المتحكم من البرنامج الذى ينفذه ويعطى رسالة خطأ تعرف المستخدم بهذا الخطأ. هناك أيضاً بعض الأوامر التي بتنفيذها تحدث مقاطعة للمتحكم ومن أمثلة ذلك الأمر reset والذي يمكن وضعه في البرنامج بحيث إذا حدثت أخطاء معينة يقوم البرنامج بعمل إعادة وضع للمتحكم لتجنب أي مواقف خطيرة قد تحدث من الاستمرار في تنفيذ البرنامج. البعض يقسم المقاطعة على حسب طبيعة الإشارة المستخدمة في المقاطعة، فهناك مثلاً مقاطعة المستوى level interrupt والتي فيها يتم وضع إشارة بمستوى معين (واحد أو صفر مثلاً) وتظل الإشارة على هذا المستوى إلى أن تقبل المقاطعة، وإذا تغيرت الإشارة عن هذا المستوى قبل قبول المقاطعة فإن المقاطعة لن تقبل. في مقابل هذا النوع هناك مقاطعة الحافة edge interrupt (الحافة الصاعدة أو النازلة لإشارة المقاطعة). في هذا النوع يكفى أن تنتقل الإشارة من الصفر إلى الواحد (الحافة الصاعدة) مثلاً لكي تقبل المقاطعة. عند حدوث هذه الحافة يتم تسجيل 1 في ماسك معين داخل المتحكم يقوم المتحكم بقراءة حالته قبل تنفيذ أي أمر، فإن وجد واحد في هذا الماسك يقوم فوراً بقبول المقاطعة. لاحظ أن هذه المقاطعة لابد أن يقبلها المتحكم وليس هناك فرصة لعدم قبوليها مثل مقاطعة المستوى السابقة. البعض يقسم المقاطعة إلى مقاطعة يمكن حجبها maskable interrupt وأخرى لا يمكن حجبها nonmaskable. في المقاطعة التي يمكن حجبها يكون قبول المقاطعة مرتبطاً بحالة علم معين داخل المتحكم، فإذا كان هذا العلم نشطاً يتم قبول المقاطعة، وإذا لم يكن هذا العلم نشطاً فلن تقبل المقاطعة، وهذا العلم يمكن التحكم فيه عن طريق المستخدم. أما المقاطعة غير القابلة للحجب فهي مقاطعة ليست مرتبطة بعلم أو خلافه وعند حدوثها لابد من قبوليها من قبل المتحكم، ولذلك يتم توصيل المقاطعة الأكثر خطورة على هذا النوع من المقاطعة حتى نضمن قبوليها وتنفيذها.

## كيف يعرف المتحكم مكان برنامج خدمة المقاطعة؟

جميع المتحكمات تحتوي مكان يسمى متوجه المقاطعة interrupt vector وهذا المكان يحتوى عنوان بداية برنامج الخدمة interrupt service routine، ISR لكل مقاطعة يمكن أن يتعامل معها هذا المتحكم. في العادة يشغل متوجه المقاطعة أول جزء من ذاكرة البرمجة لأى متحكم، وفي العادة يكون أول واحد كيلوبايت من هذه الذاكرة. عند حدوث أي مقاطعة يقفز المتحكم فوراً إلى مكان مخصص لهذه المقاطعة في متوجه المقاطعة، ليعرف منه عنوان أول أمر

في برنامج خدمة المقاطعة فيقفز إليه ويدأ في تنفيذه. في العادة ينتهي برنامج خدمة المقاطعة بالأمر RETI أى العودة من المقاطعة، حيث يتسبب تنفيذ هذا الأمر بأن يعود المتحكم إلى نفس المكان الذي خرج منه في البرنامج الأساسي.

### كيف يعود المتحكم إلى نفس المكان الذي خرج منه في البرنامج الأساسي بعد تنفيذ برنامج خدمة المقاطعة؟

يرجع الفضل في ذلك إلى ما يسمى بالمدكدة stack، وهذه المدكدة تكون عبارة عن جزء من ذاكرة البيانات أو الذاكرة العشوائية RAM يكون عنوان البداية فيه موجودة في أحد مسجلات المعالج أو المتحكم يسمى مؤشر المدكدة program counter, PC عند حدوث المقاطعة في أى وقت يكون عداد البرنامج stack pointer, SP

العنوان 0x3f	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

القيم التلقائية عند إعادة الوضع

شكل ٢-٦ مسجل الحالة في المتحكم في atmega328 حيث علم المقاطعة I هو البت رقم ٧

بعد أن ينتهي المتحكم من تنفيذ برنامج خدمة المقاطعة ISR. يقفز إلى برنامج خدمة المقاطعة RETI يقوم بسحب محتويات عداد البرنامج PC من المدكدة مرة ثانية، وبذلك يصبح الأمر الذي عليه الدور في التنفيذ هو نفس الأمر الذي كان سينفذ لو لا حدوث المقاطعة. بالطبع فإن هناك تفاصيل أكثر دقة للتعامل مع المدكدة ليس هذا هو المكان المناسب لشرحها. بعد أن عرفنا ببرنامج خدمة المقاطعة، والمدكدة، وكيفية العودة إلى نفس المكان في البرنامج الأساسي سنقدم في الجزء التالي الخطوات المرتبة التي يتبعها المتحكم عند حدوث أى مقاطعة.

### الخطوات التي ينفذها المتحكم عند حدوث أى مقاطعة

عند حدوث مقاطعة للمتحكم من أى نوع يقوم المتحكم بتنفيذ الخطوات التالية:

- ينتهي المتحكم من تنفيذ الأمر الحالى الذى يقوم بتنفيذته
- على حسب نوع المقاطعة يقفز المعالج إلى مكان محدد في متوجه المقاطعة interrupt vector ليعرف منه عنوان برنامج خدمة المقاطعة Interrupt service routine, ISR

تعرفنا عليه في الفصل الأول يحتوى عنوان الأمر الذى عليه الدور في التنفيذ في البرنامج الأساسي. بعد أن ينتهي المتحكم من تنفيذ الأمر الحالى الذى كان ينفذه أثناء حدوث المقاطعة، فإن المتحكم يقوم بدفع محتويات عداد البرنامج PC في المدكدة بمساعدة مؤشرها SP، ويقفز إلى برنامج خدمة المقاطعة RETI.

المتحكم يأخذ محتويات عداد البرنامج PC من المدكدة مرة ثانية، وبذلك يصبح

الأمر الذي عليه الدور في التنفيذ هو نفس الأمر الذي كان سينفذ لو لا حدوث المقاطعة.

بالطبع فإن هناك تفاصيل أكثر دقة للتعامل مع المدكدة ليس هذا هو المكان المناسب لشرحها.

بعد أن عرفنا ببرنامج خدمة المقاطعة، والمدكدة، وكيفية العودة إلى نفس المكان في البرنامج الأساسي سنقدم في الجزء التالي الخطوات المرتبة التي يتبعها المتحكم عند حدوث أى مقاطعة.

- ٣- لكي يقبل المتحكم أى مقاطعة لابد أن يكون علم المقاطعة I, Interrupt flag فعالا (يساوي واحد) وهو البت رقم ٧ في مسجل الحالة كما في شكل ٢-٦، ويتم ذلك قبل طلب المقاطعة.
- ٤- كل مقاطعة يكون لها علم تنشيط خاص بها وهذا العلم لابد من تنشيطة أيضا حتى تقبل هذه المقاطعة، وسترى ذلك عند الحديث عن المقاطعات المختلفة.
- ٥- إذا كانت المقاطعة ستقبل بناء على حالة العلمين السابقين، فإن المعالج يقوم بدفع عدد البرنامج program counter, PC (الذى يحتوى عنوان الأمر الذى عليه الدور فى التنفيذ فى البرنامج الأساسى) فى المكادسة .stack
- ٦- يقفز المتحكم إلى برنامج خدمة المقاطعة ISR ويعود دخوله فيه يقوم بتصرفير علم المقاطعة I ليمتنع أى مقاطعة أخرى قد تأتى و يجعل المتحكم يخرج من برنامج خدمة المقاطعة الحالية قبل إتمامها.
- ٧- يمكن للمستخدم داخل برنامج خدمة المقاطعة أن يعيد وضع علم المقاطعة بواحد ليسمه للمقاطعات الأخرى (إن حدثت) أن تخرج بعملية التنفيذ من برنامج الخدمة الحالى لخدمة مقاطعات أخرى.
- ٨- بعد أن ينتهى المتحكم من تنفيذ برنامج خدمة المقاطعة ISR الذى يكون آخر أمر فيه هو الأمر المميز RETI والذى بتنفيذته يقوم المتحكم بسحب محتويات عدد البرنامج من المكادسة مرة ثانية وبذلك تعود عملية التنفيذ إلى البرنامج الأساسى.
- ٩- يقوم المتحكم بإرجاع علم المقاطعة I إلى القيمة ١ مرة أخرى حتى يكون جاهزا لاستقبال أى مقاطعات أخرى.

### ماذا سيحدث لو أنه حدثت مقاطعات للمتحكم في نفس الوقت (بالصدفة طبعاً)؟

تحتختلف الكثير من المعالجات والمتحكمات في التعامل مع هذا الموقف فالكثير من المعالجات تعطى أولوية معينة لكل نوع من أنواع المقاطعة بحيث إذا حدث هذا الموقف بين أى مصدرى مقاطعة، فإن المقاطعة ذات الأولوية الأعلى هي التي يتم خدمتها أولاً. المتحكمات AVR أخذت هذا الأمر ببساطة ولم تعطى أولوية لأى مقاطعة سوى ترتيب وضعها في متوجه المقاطعة الذى سندرسه بعد قليل للمتحكم atmega328. فمثلاً في هذا المتوجه للمقاطعة الذى سنراه بعد قليل تأتى المقاطعة العتادية Reset في أول هذا المتوجه أو الجدول، ولذلك فهي تأخذ أعلى أولوية، فإذا جاءت مع أى مقاطعة أخرى في الجدول في نفس الوقت، فإنه يتم خدمتها أولاً. وهكذا مع المقاطعات الأخرى تكون المقاطعة التي تأتى في جدول المقاطعة أولاً هي التي لها الأولوية الأعلى.

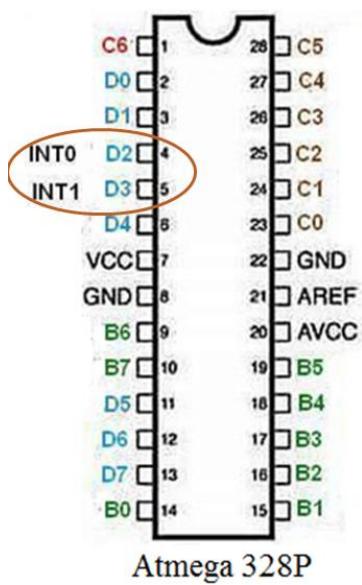
## كيف يشعر المتحكم بوجود أي طلب للمقاطعة؟

بعد أن ينتهي المتحكم (وحدة المعالجة المركزية CPU) من تنفيذ أي أمر، وقبل أن ينتقل إلى تنفيذ الأمر التالي له، فإنه يقوم بالمرور على جميع مصادر المقاطعة ليرى إن كان أحدها نشطاً (أي يتطلب المقاطعة) أم لا، فإذا وجد أي طلب للمقاطعة موجود، وجميع شروط قبول هذه المقاطعة محققة، فإنه يذهب على الفور لتنفيذ برنامج خدمة المقاطعة الخاص بها. أما إذا لم يجد أي طلبات للمقاطعة، أو هناك مقاطعة غير محققة لشروط القبول (كأن يكون علم المقاطعة الخاص بها mask يساوى صفر) فإنه ينتقل إلى تنفيذ الأمر التالي في البرنامج الأساسي الذي ينفذه.

## ٢-٦ متوجه المقاطعة Interrupt Vector

يحتوى المتحكم atmega328 على ٢٦ مقاطعة مختلفة المصدر مبينة في الجدول ١-٦ حيث تم إعطاء المقاطعات المتمدة المصدر لون خلفية واحد ليسهل تمييزها عن المقاطعات الأخرى. سنتناول في هذا الفصل المقاطعات التي

مصدرها من خارج المتحكم، أي أنها تعطى من خلال أحد أطرافه الخاصة بذلك مثل الطرفين INT0 و INT1 والمقاطعات بسبب أي تغيير في الجهد على أي طرف من أطراف المتحكم. هذه المقاطعات الخارجية هي المجموعة التالية للمقاطعة RESET مباشرة وأعطيناها اللون البمي الخفيف لتمييزها. كل واحدة من هذه المقاطعات على وجه العموم يخصه موضعين في ذاكرة البرمجة بدءاً من العنوان 0x0000 كما في الجدول وهو بمثابة متوجه المقاطعة لكل منها. بمجرد حدوث أي مقاطعة وقوتها من قبل المتحكم فإن المتحكم يقفز إلى المكان المخصص لها في جدول متوجه المقاطعة حيث يكون عنوان برنامج خدمة المقاطعة ISR الخاص بهذه المقاطعة موجوداً في هذين المكانين. فمثلاً، إذا حدثت مقاطعة على الطرف INT0، فإن المتحكم سيقفز إلى العنوان 0x0002 في ذاكرة البرمجة حيث يقرأ عنوان بداية برنامج خدمة المقاطعة ISR من هذين المكانين 0x0002 و 0x0003. باقي المقاطعات ذات المصادر الداخلية التي تأتي من الأجهزة الملحقة داخل المتحكم مثل المؤقتات والمحول التماضي الرقمي والتراسل التتابعى وغيرها سيأتى الحديث عنها في الفصول الخاصة بكل واحد من هذه الملحقات.



شكل ٣-٦ طرق المقاطعة

INT0 و INT1

### ٣-٦ المقاطعة الخارجية على الطرفين INT1 و INT0

هذان الطرفان مشتركان وظيفياً مع الطرفين D2 و D3 في البوابة D كما في شكل ٣-٦. يمكن إدخال أي مقاطعة خارجية، مثل زيادة في درجة الحرارة أو الضغط أو السرعة عن حد تشبع معين، أو اصطدام الروبوت بحائط مثلاً، من خلال هذين الخطين. كما هو موضح في جدول متوجه المقاطعة، الجدول ١-٦، فإن المقاطعة على هذين الخطين لها الأولوية التالية من حيث الترتيب بعد المقاطعة RESET التي لها أعلى أولوية على كل المقاطعات الأخرى. يتم التحكم في أداء المقاطعة من على هذين الخطين باستخدام ثلاثة مسجلات سنشرحها بالتفصيل فيما يلى.

جدول ١-٦ متوجه المقاطعة interrupt vector للمتحكم atmega328

رقم المتوجه	العنوان في ذاكرة البرمجة	مصدر المقاطعة	الوصف
1	0x0000	RESET	يمكن لهذه المقاطعة أن تحدث من الطرف reset، أو بعد تشغيل القدرة، أو بسبب دائرة مراقبة القدرة brown out، أو بسبب مؤقت كلب الحراسة
2	0x0002	INT0	طلب مقاطعة من على الطرف INT0 للشريحة
3	0x0004	INT1	طلب مقاطعة من على الطرف INT1 للشريحة
4	0x0006	PCINT0	المقاطعة رقم 0 بسبب تغير في جهد الطرف
5	0x0008	PCINT1	المقاطعة رقم 1 بسبب تغير في جهد الطرف
6	0x000A	PCINT2	المقاطعة رقم 2 بسبب تغير في جهد الطرف
7	0x000C	WDT	المقاطعة بسبب فيضان في مؤقت كلب الحراسة
8	0x000E	TIMER2_COMPA	تطابق مقارنة A في المؤقت ٢
9	0x0010	TIMER2_COMPB	تطابق مقارنة B في المؤقت ٢
10	0x0012	TIMER2_OVF	فيضان في المؤقت ٢
11	0x0014	TIMER1_CAPT	النقطاط حدث للمؤقت ١
12	0x0016	TIMER1_COMPA	تطابق مقارنة A في المؤقت ١
13	0x0018	TIMER1_COMPB	تطابق مقارنة B في المؤقت ١
14	0x001A	TIMER1_OVF	فيضان في المؤقت ١
15	0x001C	TIMERO_COMPA	تطابق مقارنة A في المؤقت ٠
16	0x001E	TIMERO_COMPB	تطابق مقارنة B في المؤقت ٠
17	0x0020	TIMERO_OVF	فيضان في المؤقت ٠
18	0x0022	SPI STC	اكتمال الإرسال التتابعى بالنظام SPI
19	0x0024	USART_RX	اكتمال الاستقبال التتابعى Rx فى النظام USART
20	0x0026	USART_UDRE	مسجل البيانات فاصل فى النظام USART
21	0x0028	USART_TX	اكتمال الإرسال التتابعى Tx فى النظام USART
22	0x002A	ADC	اكتمال التحويل فى المحول التمائى الرقمى ADC
23	0x002C	EE READY	ذاكرة EEPROM جاهزة
24	0x002E	ANALOG COMP	المقارن التمائى
25	0x0030	TWI	الواجهة مع نظام السلكين للتراسل I2C
26	0x0032	SPM READY	ذاكرة تخزين البرنامج جاهزة

**١-سجل قناع (ماسك) المقاطعة الخارجية EIMSK**

شكل ٦-٤ يبين تفاصيل هذا المسجل. كما نرى فإن هذا المسجل مكون من ٨ بتات مثل جميع المسجلات في المتحكمات AVR ذات الثمانية بتات. البتات من البت رقم ٢ حتى البت رقم ٧ غير مستخدمة.

**البت رقم ٠** خاصة بطرف المقاطعة INT0 وهي تمثل القناع أو الماسك mask الخاص بهذا الطرف. أى أنه لكي تقبل المقاطعة على الطرف INT0 فإنه لابد من وضع ١ في هذه البت، وهذا طبعاً على أن يكون علم المقاطعة العام I في مسجل حالة المتحكم قد تم وضعه بواحد هو الآخر. أما طبيعة الإشارة التي ستحدث المقاطعة من حيث هل هي إشارة مستوى أم إشارة حافة فسيتم تحديدها في المسجل التالي.

**البت رقم ١** خاصة بطرف المقاطعة INT1 وهي تمثل القناع أو الماسك mask الخاص بهذا الطرف. أى أنه لكي تقبل المقاطعة على الطرف INT1 فإنه لابد من وضع ١ في هذه البت، وهذا طبعاً على أن يكون علم المقاطعة العام I في مسجل حالة المتحكم قد تم وضعه بواحد هو الآخر. أما طبيعة الإشارة التي ستحدث المقاطعة من حيث هل هي إشارة مستوى أم إشارة حافة فسيتم تحديدها في المسجل التالي. الإسم الرسمي لهذا المسجل الذي سيستعمل في البرمجة هو EIMSK

7	6	5	4	3	2	1	0
						INT1	INT0

القيم الابتدائية عند بداية التشغيل

شكل ٦-٤ مسجل قناع المقاطعة الخارجية EIMSK

**٢-سجل تحكم المقاطعة الخارجية****External Interrupt Control Register A, EICRA**

البتات الأربع الأولي فقط من هذا المسجل هى المستخدمة والأربعة الثانية غير مستخدمة كما هو موضح في شكل ٥-٦.

**البت ٠ والبت ١:** تخصان التحكم في كيفية استشعار المقاطعة على الخط INT0 من حيث هل هي مقاطعة مستوى أم مقاطعة حافة. Interrupt Sense Control bits 0,1 وختصاراً تكتب ISCB0,1. جدول ٦-٢ يبين حالة هذه البتات (الكتابة بالخط الأسود) وكيفية استشعار المتحكم لإشارة المقاطعة المعطاة على الطرف INT0.

**البت ٢ والبت ٣:** تخصان التحكم في كيفية استشعار المقاطعة على الخط INT1 من حيث هل هي مقاطعة مستوى أم مقاطعة حافة. Interrupt Sense Control bits 0,1 وختصاراً تكتب ISCB0,1. جدول ٦-٢ يبين حالة هذه البتات (الكتابة بالخط الأزرق) وكيفية استشعار المتحكم لإشارة المقاطعة المعطاة على الطرف INT1.

7	6	5	4	3	2	1	0
ISC01	ISC00	ISC11	ISC10	ISC01	ISC00		
0	0	0	0	0	0	0	0

القيم التلقائية عند بداية التشغيل

شكل ٦-٥ مسجل تحكم المقاطعة الخارجية EICRA

الأصفار التي في أسفل المسجلات الموضحة في الشكل ٦-٤ والشكل ٦-٥ تعنى أنه عند عمل RESET للمتحكم فإن محتويات كل البتات في هذه المسجلات ستكون أصفارا.

جدول ٦-٢ برات حالة استشعار المقاطعة على الخطين INT0 و INT1

		الوصف (حالة الإشارة التي تتسبب المقاطعة)					
ISC01 ISC11	ISC00 ISC10						
0	0	يتم طلب المقاطعة على الخط INT0 باشارة منخفضة المستوى (صفر وتظل صفر الى أن تقبل)					
0	0	يتم طلب المقاطعة على الخط INT1 باشارة منخفضة المستوى (صفر وتظل صفر الى أن تقبل)					
0	1	أى تغير منطقى على الخط INT0 (صفر إلى واحد، أو واحد إلى صفر) يتسبب في المقاطعة					
0	1	أى تغير منطقى على الخط INT1 (صفر إلى واحد، أو واحد إلى صفر) يتسبب في المقاطعة					
1	0	الحافة الهابطة (واحد إلى صفر) على الخط INT0 تتسبب في المقاطعة					
1	0	الحافة الهابطة (واحد إلى صفر) على الخط INT1 تتسبب في المقاطعة					
1	1	الحافة الصاعدة (صفر إلى واحد) على الخط INT0 تتسبب في المقاطعة					
1	1	الحافة الصاعدة (صفر إلى واحد) على الخط INT1 تتسبب في المقاطعة					

الأسود خاص بالخط INT0

الأزرق خاص بالخط INT1

### ٣-مسجل الأعلام للمقاطعة الخارجية EIFR External Interrupt Flag Register, EIFR

7	6	5	4	3	2	1	0
						INTF1	INTF0
						0	0

القيم التلقائية عند بداية التشغيل

شكل ٦-٦ مسجل الأعلام للمقاطعة الخارجية EIFR

البتات من رقم ٢ إلى رقم ٧ غير مستخدمة.

**البت رقم ٠:** تمثل علم مقاطعة للخط INT0. إذا كان بتات استشعار إشارة المقاطعة ISC00 و ISC01 في الجدول ٦-٢ (بالخط الأسود) تساوى ٠٠، مما يعني أن هذه المقاطعة ستكون مقاطعة مستوى، أي أن الطرف

لابد أن يساوى صفر وأن يبقى صفر حتى تقبل المقاطعة، في هذه الحالة يكون العلم INTF0 في شكل ٦-٦ يساوى صفر، ولن يكون له أى تأثير على المقاطعة. المشكلة هي في الحالات الثلاث المتبقية في الجدول ٢-٦، أى إذا كان الطرف INT0 حساساً لأى تغير منطقى أو لأى حافة، في هذه الحالة بمجرد حدوث هذا التغير المنطقى أو الحافة الصاعدة أو النازلة) فإن العلم INTF0 يصبح واحد، بحيث إذا كان هذا العلم واحد، وكان القناع INT0 في مسجل الأقنعة يساوى واحد، وكان علم المقاطعة العام I يساوى واحد، فإن المقاطعة ستقبل، وتنتقل عملية التنفيذ إلى برنامج ISR. أى أن البت رقم 0 في هذا المسجل تعمل بمثابة ماسك latch يسجل حالة الخط INT0 عندما يكون حساساً لأى تغير منطقى أو لأى حافة.

**البت رقم 1:** تقلل علم مقاطعة للخط INT1. إذا كان بتات استشعار إشارة المقاطعة ISC10 و ISC11 في الجدول ٢-٦ (بالخط الأزرق) تساوى ٠٠، مما يعني أن هذه المقاطعة ستكون مقاطعة مستوى، أى أن الطرف 1 لابد أن يساوى صفر وأن يبقى صفر حتى تقبل المقاطعة، في هذه الحالة يكون العلم INTF1 في شكل ٦-٦ يساوى صفر، ولن يكون له أى تأثير على المقاطعة. المشكلة هي في الحالات الثلاث المتبقية في الجدول ٢-٦، أى إذا كان الطرف 1 INT1 حساساً لأى تغير منطقى أو لأى حافة، في هذه الحالة بمجرد حدوث هذا التغير المنطقى أو الحافة الصاعدة أو النازلة) فإن العلم INTF1 يصبح واحد، بحيث إذا كان هذا العلم واحد، وكان القناع INT1 في مسجل الأقنعة يساوى واحد، وكان علم المقاطعة العام I يساوى واحد، فإن المقاطعة ستقبل، وتنتقل عملية التنفيذ إلى برنامج ISR. أى أن البت رقم 1 في هذا المسجل تعمل بمثابة ماسك latch يسجل حالة الخط INT1 عندما يكون حساساً لأى تغير منطقى أو لأى حافة.

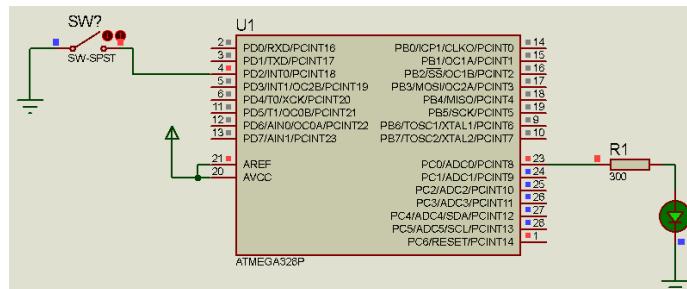
إذن ملخص ما سبق هو أن خطى المقاطعة الخارجية INT0 و INT1 يتم التحكم فيما من خلال ثلاث مسجلات: الأول يحتوى قناع لكل مقاطعة وهذا القناع لابد أن يكون واحد حتى تقبل المقاطعة على الخط المقابل، والمسجل الثاني يحتوى اثنين بت لك خط مقاطعة يتحكمان في طريقة استشعار إشارة المقاطعة هل هي حساسة للمستوى أم للحافة الصاعدة أم النازلة، وأما المسجل الثالث فيحتوى علم لكل مقاطعة يسجل فيه واحد إذا كانت المقاطعة حساسة للتغير المنطقى أو للحافة الصاعدة أو النازلة.

### مثال على المقاطعة:

البرنامج التالي يدور في الحلقة المغلقة while(1) ولا يعمل شيء، وعند حدوث المقاطعة على الخط INT0 فإن المتحكم يقفر إلى برنامج خدمة المقاطعة (INT0\_vect). برنامج خدمة المقاطعة سيعكس حالة بت المخرج PC0 الموصلة على دايدود ضوئي LED عند حدوث أى مقاطعة على هذا الطرف.

```
/*
* Interrupt1.c
*
* Created: 6/30/2017 6:57:28 AM
* Author : Mohamed Eladawy
*/
#include <avr/io.h>
#include <avr/interrupt.h>

int main(void)
{
    DDRD &=~(1<<DDD2);
```



شكل ٧-٦ استخدام المقاطعة INT0 لعكس الدايوه الضوئي

// طرف المقاطعة يجب أن يكون دخل حتى //  
يقبل المقاطعة عليه

عند حدوث تغير في وضع المفتاح

```
PORTD |=(1<<PORTD2); // تفعيل مقاومة الجذب على هذا الطرف
EICRA |=(1<<ISC00); // تفعيل المقاطعة لعمل عند أى تغير منطقى
EIMSK |=(1 << INT0); // تفعيل قناع المقاطعة
sei(); // تفعيل علم المقاطعة العام
DDRC =0xFF; // البوابة سى بوابة إخراج
PORTC =0x01; // إضاءة الدايوه
while (1)
{
}
```

ISR (INT0\_vect) // بداية برنامج خدمة المقاطعة

```
{
    PORTC ^= (1<<0);
```

عكس إضافة daiod الضوئي //

}

نلاحظ في هذا البرنامج تضمين الملف المكتبي القياسي Interrupt.h بالأمر #include <avr/interrupt.h> وهو ملف مكتبي قياسي، أى أنه من إعداد الأتمل استديو، وذلك حتى يمكن التعامل مع المقاطعة. يجب أن يكون طرف البوابة D المستخدم في المقاطعة INT0 وهو الطرف PD2 طرف دخل وتفعيل مقاومة الجذب الداخلية عليه عن طريق إخراج واحد عليه، وهذا ما يفعله أول أمرین داخل البرنامج الأساسي main. بعد ذلك تم وضع بثات استشعار الإشارة على الطرف INT0 لتساوي 01، وهذا الوضع يجعل الطرف INT0 حساس لأى تغيير منطقى عليه. لاحظ أننا استخدمنا الأمر ;EICRA | =1<>ISC00 ليجعل البت رقم صفر تساوى واحد، ولم نعمل شيء في البت رقم 1 لأنها تكون صفر تلقائيا مع بداية التشغيل. هذا الأمر يعني إزاحة 1 في البت ISC00 في مسجل التحكم في المقاطعة الخارجية EICRA. بعد ذلك تم وضع قناع المقاطعة INT0 يساوى واحد باستخدام الأمر |=EIMSK <>(1)(); وهو أحد الأوامر القياسية التي سنستخدمها باستمرار لتفعيل هذا العلم. بذلك أصبح الطرف INT0 جاهزا لاستقبال أي مقاطعة. لذلك سيدور البرنامج في الحلقة المغلقة () while بدون أن يعمل أي شيء، إلى أن يحدث أي تغيير في حالة المفتاح الموصى على الطرف INT0، حيث عندها سيقفز المتحكم إلى برنامج خدمة المقاطعة ISR لينفذه، وهذا البرنامج عبارة عن أمر واحد يعكس حالة طرف البوابة PC0 الموصى على daiod الضوئي، وبعدها يعود ليدور في الحلقة المغلقة. حاول اللعب بهذا البرنامج عن طريق تغيير الحالات المختلفة لاستشعار إشارة المقاطعة ورؤية تأثيرها، وبنفس الطريقة حاول اللعب أيضا بطرف المقاطعة INTR1.

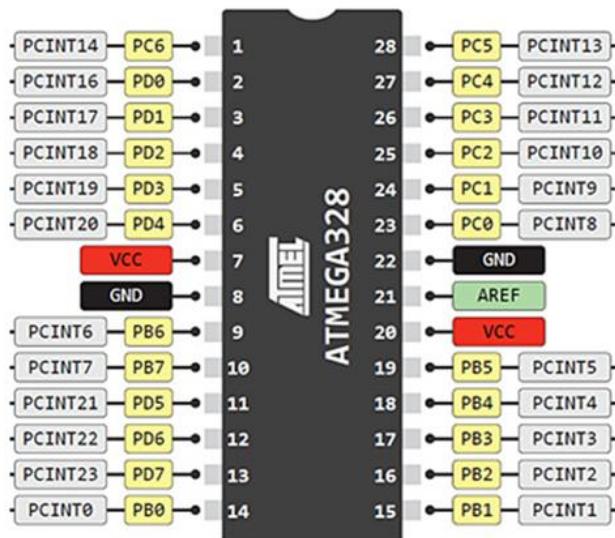
## ٦-٤ المقاطعة بسبب التغيير على أي طرف Pin Change

هناك نوع آخر من المقاطعة الخارجية بالإضافة إلى المقاطعة على الطرفين INT0 و INT1 وهو المقاطعة عند حدوث أي تغير في الجهد المنطقي لأطراف بوابات الإدخال والإخراج الثلاثة PC و PB و PD. لذلك فإن هناك ثلاثة مججهات لهذا النوع من المقاطعة موضحة في جدول متوجه المقاطعة الموضح في الجدول ٦-١، وهذه المججهات لها الأولوية التالية مباشرة للمقاطعة الخارجية INT1، وهي كالتالي:

١- المقاطعة PCINT0 ورقم المتجه الخاص بها هو ٤ كما في جدول ٦-١، وعنوان المتجه الخاص بها في ذاكرة البرمجة هو ٠x0006، وهذه المقاطعة تنشط في حالة حدوث أى تغير منطقى على الأطراف الشمانية للبوابة PB. أى أنه يمكن القول أن هذه المقاطعة، PCINT0، لها ثمانية مصادر وهى الأطراف ٠PB0 حتى PB7. في العادة تأخذ هذه المصادر الأرقام PCINT0 حتى PCINT7.

٢- المقاطعة PCINT1 ورقم المتجه الخاص بها هو ٥ كما في جدول ٦-١، وعنوان المتجه الخاص بها في ذاكرة البرمجة هو ٠x0008، وهذه المقاطعة تنشط في حالة حدوث أى تغير منطقى على الأطراف السبعة للبوابة PC. أى أنه يمكن القول أن هذه المقاطعة، PCINT1، لها سبعة مصادر وهى الأطراف ٠PC0 حتى PC6. في العادة تأخذ هذه المصادر الأرقام PCINT8 حتى PCINT14.

٣- المقاطعة PCINT2 ورقم المتجه الخاص بها هو ٦ كما في جدول ٦-١، وعنوان المتجه الخاص بها في ذاكرة البرمجة هو ٠x000A، وهذه المقاطعة تنشط في حالة حدوث أى تغير منطقى على الأطراف الشمانية للبوابة PD. أى أنه يمكن القول أن هذه المقاطعة، PCINT2، لها ثمانية مصادر وهى الأطراف ٠PD0 حتى PD7. في العادة تأخذ هذه المصادر الأرقام PCINT16 حتى PCINT23.



شكل ٦-٨-٦ البوابات الثلاثة التي يمكن مقاطعة المتحكم

من خلال أى تغير على أطرافها

#### ١-مسجل التحكم في مقاطعة تغير الأطراف

#### Pin Change Interrupt Control Register, PCICR

شكل ٦-٩-٦ يبين بنات هذا المسجل، حيث نلاحظ استخدام ثلاث برات فقط منه، وكل برت تمثل القناع أو بت التنشيط لكل واحدة من هذه المقاطعات.

7	6	5	4	3	2	1	0		
							PCIE2	PCIE1	PCIE0
							0	0	0

القيم التلقائية عند بداية التشغيل

شكل ٦-٩ مسجل التحكم في مقاطعة تغير الأطراف PCICR

**البت ٠:** تنشيط مقاطعة تغير الأطراف رقم ٠، Pin Change Interrupt Enable PCIE0، حيث يوضع هذه البت بواحد وإذا كان علم المقاطعة العام I يساوى واحد، فإن أي تغيير على الأطراف PB0 إلى PB7 سيتسبب في تفعيل هذه المقاطعة. يمكن إخماد أو تنشيط كل واحد من هذه المقاطعات الثمانية منفردا باستخدام مسجل الأقنية .PCMSK0

**البت ١:** تنشيط مقاطعة تغير الأطراف رقم ١، Pin Change Interrupt Enable PCIE1، حيث يوضع هذه البت بواحد وإذا كان علم المقاطعة العام I يساوى واحد، فإن أي تغيير على الأطراف PC0 إلى PC6 سيتسبب في تفعيل هذه المقاطعة. يمكن إخماد أو تنشيط كل واحد من هذه المقاطعات الثمانية منفردا باستخدام مسجل الأقنية .PCMSK1

**البت ٢:** تنشيط مقاطعة تغير الأطراف رقم ٢، Pin Change Interrupt Enable PCIE2، حيث يوضع هذه البت بواحد وإذا كان علم المقاطعة العام I يساوى واحد، فإن أي تغيير على الأطراف PD0 إلى PD7 سيتسبب في تفعيل هذه المقاطعة. يمكن إخماد أو تنشيط كل واحد من هذه المقاطعات الثمانية منفردا باستخدام مسجل الأقنية .PCMSK2

**البّات من ٣ حتّى ٧:** غير مستخدمة.

## ٢-مسجل أعلام مقاطعة تغير الأطراف PCIFR

شكل ٦-١٠ يبين بّات هذا المسجل، حيث نلاحظ استخدام ثلاثة بّات فقط منه، وكل بّت تمثل العلم الخاص بكل واحدة من هذه المقاطعات.

7	6	5	4	3	2	1	0		
							PCIF2	PCIF1	PCIF0
							0	0	0

القيم التلقائية عند بداية التشغيل

شكل ٦-١٠ مسجل أعلام مقاطعة تغير الأطراف PCIFR

**البت 0:** عند حدوث أى تغير منطقى على أى واحد من المصادر PCINT0 أو PB0 حتى PCINT7 فإن العلم PCIF0 سيصبح واحد، وإذا كان علم المقاطعة العام I يساوى واحد، وقناع هذه المقاطعة PCIE0 في المسجل السابق يساوى واحد، فإن المتحكم سيقفز إلى برنامج خدمة هذه المقاطعة ISR. بمجرد الانتهاء من برنامج خدمة المقاطعة يتم تصفير هذا العلم تلقائيا. يمكننا القول بأن هذا العلم بمثابة ماسك latch يمسك بالتغيير الذى حدث على هذه الأطراف إلى أن تقبل المقاطعة، وذلك لأن هذا التغيير يكون في العادة حافة صاعدة أو نازلة.

**البت 1:** عند حدوث أى تغير منطقى على أى واحد من المصادر PC0 أو PC6 حتى PCINT8 أو PCINT14 فإن العلم PCIF1 سيصبح واحد، وإذا كان علم المقاطعة العام I يساوى واحد، وقناع هذه المقاطعة PCIE1 في المسجل السابق يساوى واحد، فإن المتحكم سيقفز إلى برنامج خدمة هذه المقاطعة ISR. بمجرد الانتهاء من برنامج خدمة المقاطعة يتم تصفير هذا العلم تلقائيا. يمكننا القول بأن هذا العلم بمثابة ماسك latch يمسك بالتغيير الذى حدث على هذه الأطراف إلى أن تقبل المقاطعة، وذلك لأن هذا التغيير يكون في العادة حافة صاعدة أو نازلة.

**البت 2:** عند حدوث أى تغير منطقى على أى واحد من المصادر PD0 أو PD7 حتى PCINT16 أو PCINT23 فإن العلم PCIF2 سيصبح واحد، وإذا كان علم المقاطعة العام I يساوى واحد، وقناع هذه المقاطعة PCIE2 في المسجل السابق يساوى واحد، فإن المتحكم سيقفز إلى برنامج خدمة هذه المقاطعة ISR. بمجرد الانتهاء من برنامج خدمة المقاطعة يتم تصفير هذا العلم تلقائيا. يمكننا القول بأن هذا العلم بمثابة ماسك latch يمسك بالتغيير الذى حدث على هذه الأطراف إلى أن تقبل المقاطعة، وذلك لأن هذا التغيير يكون في العادة حافة صاعدة أو نازلة.

**البتابات من 3 حتى 7:** غير مستخدمة.

### ٣-مسجل الأقنية مقاطعة تغير الأطراف ٠، PCMSK0

شكل ١١-٦ يبين ببات هذا المسجل، حيث نلاحظ أن كل ببات هذا المسجل مستخدمة. كل ببت من ببات هذا المسجل تمثل قناعا يمكن به تنشيط أو إخماد مقاطعة التغيير على الطرف المقابل PCINT0 حتى PCINT7 كما في الشكل.

### ٤-مسجل الأقنية مقاطعة تغير الأطراف ١، PCMSK1

شكل ١٢-٦ يبين ببات هذا المسجل، حيث نلاحظ أن كل ببات هذا المسجل مستخدمة فيما عدا البت الأخيرة فقط حيث أن البوابة C في هذا المتحكم تحتوى ٨ ببات أو ٨ خطوط فقط. كل ببت من ببات هذا المسجل تمثل قناعا يمكن به تنشيط أو إخماد مقاطعة التغيير على الطرف المقابل PCINT8 حتى PCINT14 كما في الشكل.

## ٥-مسجل الأقنعة مقاطعة تغير الأطراف 2، PCMSK2

شكل ٦-١٣ يبين ببات هذا المسجل، حيث نلاحظ أن كل ببات هذا المسجل مستخدمة. كل ببت من ببات هذا المسجل تمثل قناعا يمكن به تنشيط أو إخماد مقاطعة التغيير على الطرف المقابل PCINT16 حتى PCINT23 كما في الشكل.

لاحظ أن كل هذه المسجلات تكون أصفارا عند بداية تشغيل المتحكم .RESET

البت	7	6	5	4	3	2	1	0
	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
	R/W							
	0	0	0	0	0	0	0	0

شكل ٦-١١ مسجل الأقنعة المنفردة للمقاطعات PCINT7 حتى PCINT0

البت	7	6	5	4	3	2	1	0
	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	0	0	0	0	0	0	0	0

شكل ٦-١٢ مسجل الأقنعة المنفردة للمقاطعات PCINT14 حتى PCINT8

البت	7	6	5	4	3	2	1	0
	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16
	R/W							
	0	0	0	0	0	0	0	0

شكل ٦-١٣ مسجل الأقنعة المنفردة للمقاطعات PCINT23 حتى PCINT16

من عيوب المقاطعة بسبب التغيير أن كل واحد من الثلاث مقاطعات الأساسية PCINT0 و PCINT1 و PCINT2 يتبعه ثمانية مقاطعات، وعند تنشيط أي عدد من هذه المقاطعات فإنه عند حدوث مقاطعة على أي طرف من هذه الأطراف النشطة، فإنك لن تعرف من أين أتت هذه المقاطعة. فمثلا لو أننا نشطنا المقاطعة INT0 وجعلنا الأقنعة الشمانية المصاحبة لها على الأطراف PB0 حتى PB7 نشطة كلها، فإنه عند إحداث مقاطعة على أي واحد من هذه الأطراف النشطة لن نعرف ما هو الطرف الذي حدثت عليه هذه المقاطعة حيث أن هناك برنامج خدمة مقاطعة واحد ISR من خلال متوجه مقاطعة واحد يتم تنفيذه لهذه المقاطعات الشمانية مجتمعة.

**مثال على مقاطعة التغير في الأطراف:**

في هذا البرنامج سنقوم بتنشيط مقاطعات التغيير على الأطراف الثلاثة من البوابة PB، وسنحصل على كل منها مفتاح لإحداث هذا التغيير، وسنجري أنه عند إحداث التغيير على طرف من الأطراف الثلاث، فإن المتحكم سيقفز إلى نفس برنامج خدمة المقاطعة، الذي يقوم بتغيير حالة الدايوهات الضوئية الثلاث الموصولة على الخطوط الثلاثة الأولى من البوابة PC كما في شكل ٦-١٤.

```
/*
* Interrupt2.c
*
* Created: 7/1/2017 7:03:22 AM
* Author : Mohamed Eladawy
*/
#include <avr/io.h>
#include <avr/interrupt.h>
int main(void)
{
    DDRB &= ~((1 << DDB0) | (1 << DDB1) | (1 << DDB2));
    //Clear the PB0, PB1, PB2, (PCINT0, PCINT1, PCINT2 pin) are now inputs
    PORTB |= ((1 << PORTB0) | (1 << PORTB1) | (1 << PORTB2));
    //turn On the Pull-up PB0, PB1 and PB2
    PCICR |= (1 << PCIE0); //set PCIE0 to enable PCINT0
    PCMSK0 |= (1 << PCINT0)|(1<<PCINT1)|(1<<PCINT2);
    //set PCINT0, PCINT1, and PCINT2 to trigger an interrupt on state change
    sei(); // turn on Global interrupt
    DDRC=0xFF;
    PORTC=0xff;

    while (1)
    {
    }
}
```

```

}

ISR (PCINT0_vect)
{
    PORTC ^= (1<<PORTC0); //toggle PC0
    PORTC ^= (1<<PORTC1); //toggle PC1
    PORTC ^= (1<<PORTC2); //toggle PC2
}

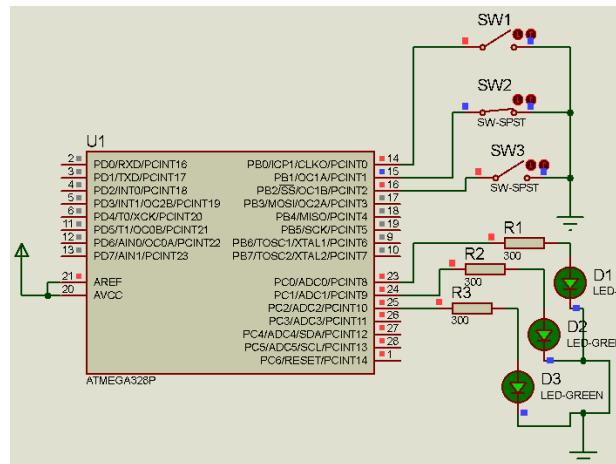
```

الآن لكي نعرف بالضبط ما هو مصدر المقاطعة الذي تسبب في الذهاب إلى برنامج خدمة المقاطعة، لابد من إجراء بعض الخطوات البرمجية في برنامج خدمة المقاطعة بحيث يمكن من خلالها تحديد المصدر الذي أحدث هذه المقاطعة.

سنرى ذلك في المثال التالي:

```

/*
 * Interrupt3.c
 *
 * Created: 7/2/2017 1:07:33 PM
 * Author : Mohamed Eladawy
 */
#include <avr/io.h>
#include <avr/interrupt.h>
volatile int portbhistry = 0xFF; // global variable initialized as 0xFF
int main(void)
{
    DDRB &= ~((1 << DDB0) | (1 << DDB1) | (1 << DDB2));
    //Clear the PB0, PB1, PB2, (PCINT0, PCINT1, PCINT2 pin) are now inputs
    PORTB |= ((1 << PORTB0) | (1 << PORTB1) | (1 << PORTB2));
    //turn On the Pull-up PB0, PB1 and PB2
    PCICR |= (1 << PCIE0); //set PCIE0 to enable PCINT0
    PCMSK0 |= (1 << PCINT0)|(1<<PCINT1)|(1<<PCINT2);
    //set PCINT0, PCINT1, and PCINT2 to trigger an interrupt on state change
}
```



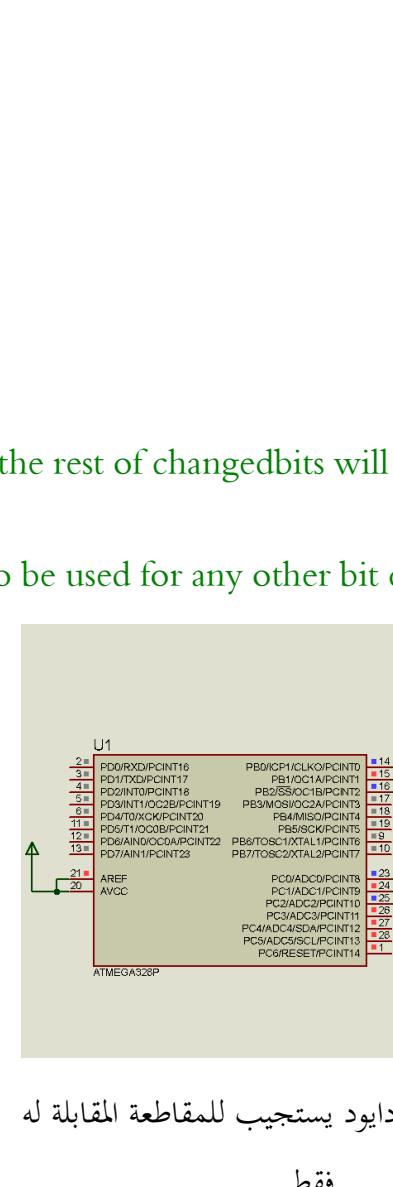
شكل ٦-٤ المقاطعات الثلاث تذهب لنفس برنامج خدمة المقاطعة

```

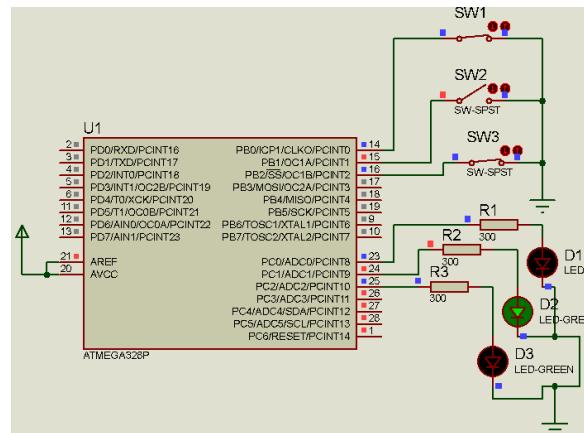
sei(); // turn on Global interrupt
DDRC=0xFF;
PORTC=0xff;
while (1)
{
}

ISR (PCINT0_vect)
{
    int changedbits;
    changedbits = PINB ^ portbhistory;
    //the changed bit only will be 1 and the rest of changedbits will be zeros
    portbhistory = PINB;
    // update the pin history of port B, to be used for any other bit change
    if(changedbits & (1 << PINB0))
    {
        PORTC ^=(1<<0);
    }
    //toggle PC0
    if(changedbits & (1 << PINB1))
    {
        PORTC ^= (1<<1);
    }
    //toggle PC1
    if(changedbits & (1 << PINB2))
    {
        PORTC ^= (1<<2); //toggle PC2
    }
}

```



٦-١٥ كل دايدو يستجيب للمقاطعة المقابله له فقط



شكل ٦-١٥ كل دايوود يستجيب للمقاطعة المقابلة له فقط

نلاحظ في هذا البرنامج أن البرنامج الأساسي هو نفسه تقريباً البرنامج الأساسي السابق، الذي ينشط الثلاث مقاطعات الأولى للتغيير على أطراف البوابة PB، مع توصيل ثلاث مفاتيح من خلالها لإحداث المقاطعة. بعد ذلك تم تعريف البوابة PC لتكون بوابة إخراج موصل عليها ثلاث دايوdas ضوئية. لقد تم إضافة الأمر `volatile int portbhitory = 0xFF;` في بوابة PC التي يعرف المتغير `portbhitory` ليكون متغير عام صحيح وقيمتة هي FF، ويمكن رؤيته في برنامج خدمة المقاطعة ISR. الجديد في هذا البرنامج موجود فقط في برنامج خدمة المقاطعة. يبدأ برنامج خدمة المقاطعة بالأمر: `int changedbits;` الذي يعرف متغيراً محلياً صحيحاً اسمه `changedbits` يمكن رؤيته فقط في برنامج خدمة المقاطعة. الفكرة هنا تتلخص في أننا نحتفظ بقيمة محتويات البوابة B في المتغير `portbhitory` الذي بدأ بالقيمة FF، ثم نقرأ محتويات البوابة B الحالية، ونقوم بعمل عملية إكس مع القيمة المقرورة من البوابة B مع قيمتها السابقة المخزنة في المتغير `portbhitory`. نتيجة هذه العملية ستكون أن البتات التي تغيرت في البوابة B بعد قراءتها هي فقط التي ستكون قيمتها وحيدة. يمكن بعد ذلك باستخدام الأمر الشرطي if معرفة أي الأطراف هي التي تغيرت وبذلك يمكن أخذ الفعل المقابل له كما في البرنامج السابق. فمثلاً الأمر:

```
if(changedbits & (1 << PINB0))
```

```
{
```

```
    PORTC ^= (1 << 0); //toggle PC0
```

```
}
```

يسأل هل البت الذي تغيرت هي البت PB0، عن طريق إجراء عملية الآند على المتغير `changedbits` مع الرقم 00000001 وهو الرقم 1 بعد إزاحته يساراً بمقدار صفر من البتات. فإذا كانت نتيجة الآند تساوى 1 سيدخل في قوس الأمر الشرطي ويعكس الطرف PC0 من خلال الأمر: `PORTC ^= (1 << 0);` وإذا لم تكن نتيجة الآند واحد، ينتقل المتحكم إلى الأمر الشرطي التالي ليسأل هل حدث التغيير على الطرف PB2 أم لا. وبنفس الطريقة تستمر الاختبارات على الأطراف النشطة من المقاطعة التي حدثت على أطراف البوابة النشطة. بذلك يمكن بناء الدائرة كما في شكل ١٥-٦ وتنفيذها باستخدام البرنامج السابق لنرى أنه مع تفعيل أي مقاطعة، فإن الدايو الضوئي المقابل لهذه المقاطعة فقط هو الذي سيتغير. لاحظ أن الدائرة التي في شكل ١٥-٦ هي نفسها دائرة الشكل ١٤-٦ مع تحميل المتحكم بالبرنامج الجديد فقط.

## ٦-٥ مشروع جراج سيارات

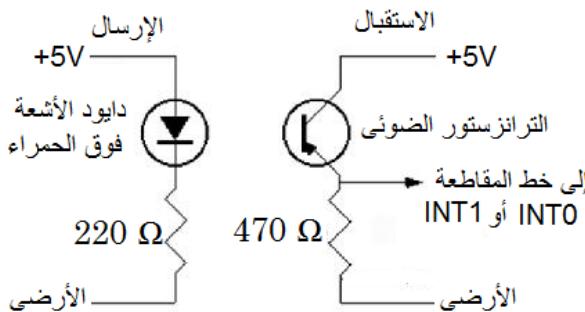
**المطلوب من المشروع:** سنفترض في هذا المشروع وجود جراج سيارات والمطلوب هو حساب عدد السيارات الداخلة، والخارجية، وعدد السيارات الموجودة بالفعل داخل الجراج وعرضها على شاشة LCD. سنفترض أن سعة الجراج هي أى عدد يتم افتراضه وسنضعه في البرنامج يساوى خمس سيارات لتسهيل اختبار البرنامج. طالما أن عدد السيارات داخل الجراج أقل من سعة الجراج سنضيء لمبة خضراء في مدخل الجراج. إذا زاد عدد السيارات الفعلى داخل الجراج عن السعة المطلوبة سنضيء لمبة حمراء مع وضع عبارة "تحطى السعة" أو "Over Capacity".

**تنفيذ الدائرة:** سنستخدم خط المقاطعة INT0 لاستشعار السيارات الداخلة، وخط المقاطعة INT1 لاستشعار السيارات الخارجية. يبين شكل ١٦-٦ الدائرة المستخدمة لاستشعار السيارات الداخلة والخارجية، ولذلك سيتم توصيل هذه الدائرة على كل من خط المقاطعة INT0 و INT1. تتكون الدائرة من دايويد مشع للأشعة فوق الحمراء infrared emitting diode, IRLED ضوئى phototransistor، وهو عبارة عن ترانزستور مفتوح القاعدة كما نرى في الشكل. عند سقوط الضوء على القاعدة يصبح الترانزستور ON وير التيار من المجمع للمشع، وفي هذه الحالة يكون الجهد الداخل إلى خط المقاطعة

على (٥ فولت) حيث أن الترانزستور يكون short circuit.

في عدم وجود ضوء ساقط على قاعدة الترانزستور، يكون الترانزستور OFF ولن يمر تيار فيه، وسيكون الجهد الموصى على طرف المقاطعة منخفض (صفر).

لاحظ أنه عند توصيل هذه الدائرة على طرف المقاطعة INT0 و INT1 لن تكون بحاجة إلى تفعيل المقاطعة على (واحد). عند مرور السيارة الداخلية



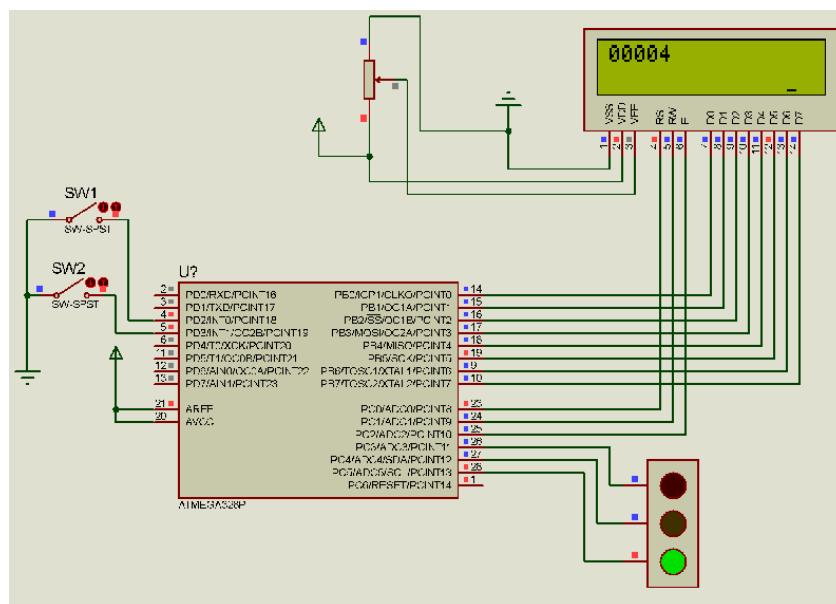
شكل ١٦-٦ استشعار السيارات الداخلة أو الخارجة باستخدام الأشعة فوق الحمراء (مرسل ومستقبل)

للجراج بين الدايويد والترانزستور سينزل جهد طرف المقاطعة من الواحد للصفر نتيجة حجب الضوء، وسيظل الجهد منخفضاً طالما أن السيارة موجودة بين مصدر الأشعة والترانزستور. بمجرد خروج السيارة من بين مصدر الأشعة والترانزستور سيرتفع جهد المقاطعة من الصفر إلى الواحد، وهنا سيعتبر النظام أن السيارة دخلت الجراج، ولذلك سيجمع واحد على عداد السيارات الموجودة داخل الجراج. لذلك فإن خط المقاطعة س يجعله حساس للحافة الصاعدة (صفر إلى واحد)

حتى لا يأخذ السيارة في الحسبان إلا بعد عبورها من منطقة الأشعة. بنفس الطريقة أيضا سنوصل مثل هذه الدائرة على طرف المقاطعة INT1 بحيث عند خروج السيارة من منطقة الأشعة (تغير جهد طرف المقاطعة من الصفر للواحد) فإنه سيتم إنفاس عدد السيارات داخل الجراج بمقدار واحد.

لكى نختبر الدائرة على برنامج البروتس فقد مثلنا نظام الأشعة السابق ذكره بمفتاح تم توصيله على الطرف INT0 وآخر تم توصيله على الطرف INT1 مع جعل طرف المقاطعة حساساً للحافة الصاعدة.

شكل ١٧-٦ يبين الدائرة التي تم بناؤها في البروتس حيث تم توصيل مفاتيحين على الخطين INT0 و INT1 لتمثيل السيارات الداخلة والخارجة، وتم توصيل الشاشة LCD على البوابة B، مع استخدام الخطوط الثلاث الأولى من البوابة C كخطوط تحكم في الشاشة، والخطوط الثلاث التالية لها لإنارة المرور بحيث يوضح النور الأخضر وجود أماكن شاغرة داخل الجراج، والنور الأحمر ليدل على أن عدد السيارات داخل الجراج قد تعدد الحد الأقصى ولم يعد هناك أماكن شاغرة.



\* Created: 7/6/2017 7:51:10 PM

\* Author : Mohamed Eladawy

\*/

```
#define F_CPU 1000000ul
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "LCDlib.h"

volatile int Capacity = 0x05; // max no. of cars as 0x0F
volatile int i = 0x00; // current no. of cars is initially 0x00

int main(void)
{
    DDRD &=~((1<<DDD2)|(1<<DDD3));
    PORTD |=((1<<PORTD2)|(1<<PORTD3));
    EICRA |=((1<<ISC00)|(1<<ISC01)|(1<<ISC10)|(1<<ISC11));
    EIMSK |=((1 << INT0)|(1 << INT1));
    sei();
    DDRC =0xFF;
    PORTC =0x01;
    LCD_Init();

    while (1)
    {
    }

    ISR (INT0_vect)
    {
        i++;
        if (i<= Capacity)
        {
    }
}
```

```

PORTC |=(1<<PORTC5); //green light ON
PORTC &= ~(1<<PORTC3); //Red light off
LCD_GoToLineOne();
LCD_DisplayNumber(i);
}
else if (i>Capacity)
{
    PORTC |=(1<<PORTC3); //Red light OFF
    PORTC &= ~(1<<PORTC5); // Green light Off
    LCD_GoToLineOne();
    LCD_DisplayNumber(i);
    LCD_GoToLineTwo();
    LCD_DisplayString("Over Capacity");
}
ISR (INT1_vect)
{
    i--;
    if (i<= Capacity)
    {
        PORTC |=(1<<PORTC5);
        PORTC &= ~(1<<PORTC3);
        LCD_GoToLineOne();
        LCD_DisplayNumber(i);
        LCD_GoToLineTwo();
        LCD_DisplayString("      ");
    }
    else if (i>Capacity)
    {
        PORTC |=(1<<PORTC3);
    }
}

```

```

PORTC &= ~(1 << PORTC5);
LCD_GoToLineOne();
LCD_DisplayNumber(i);
LCD_GoToLineTwo();
LCD_DisplayString("Over Capacity");
}
}

```

لاحظ كيف تم تضمين برنامج LCDlib.h حتى يمكن استخدام دواله المختلفة، ويرجى الرجوع في ذلك إلى الفصل ٥ الذي وضع كل هذه الدوال بالتفصيل، وكيفية إضافة مثل هذه البرامج. لاحظ أن البرنامج C يحتوى تعريف البوابة B لتمثل البيانات للشاشة LCD والبتات الثلاثة الأولى من البوابة C كخطوط تحكم.

## ملخص الفصل

بعد التعريف بمعنى المقاطعة وأنواعها وتصنيفاتها المختلفة ركز الفصل على المقاطعة الخارجية من على الطرفين INT0 و INT1 للشريحة atmega328. تم شرح طريقة تفعيل كل من المقاطعتين وأعلامهما من خلال شرح المسجلات المستخدمة في ذلك. بعد ذلك انتقل الحديث إلى شرح المقاطعة عند حدوث أي تغير في أطراف بوابات المتحكم الثلاثة من خلال شرح المسجلات المصاحبة لها. في الختام تم تقديم مثال تطبيقي متكملاً لمراجعة سيارات.

## الفصل ٧

### المحول التماشى الرقمي

### Analog to Digital Converter, ADC

**العناوين المضيئة فى هذا الفصل:**

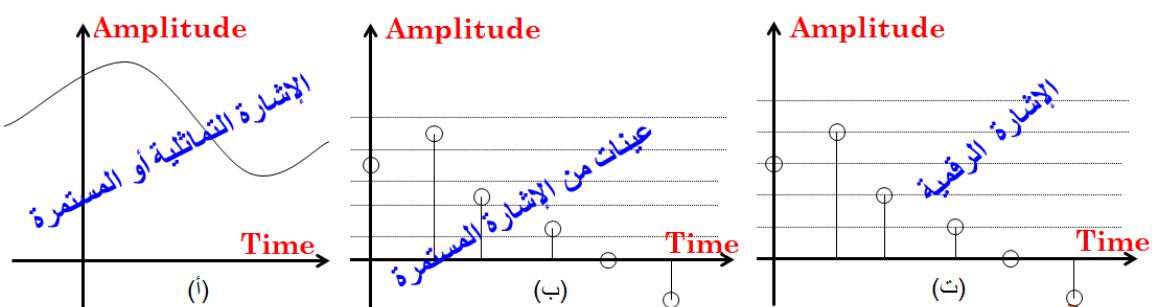
- ١- خصائص المحول التماشى الرقمى فى هذا المتحكم
- ٢- مسجل بيانات المحول التماشى الرقمى
- ٣- مسجل الاختيار من متعددات المداخل
- ٤- مسجل التحكم والحالة A
- ٥- مسجل التحكم والحالة B
- ٦- أمثلة على تشغيل المحول التماشى الرقمى
- ٧- مشروع تطبيقى
- ٨- مقترن مشروع تطبيقى للتنفيذ

## ١-٧ مقدمة

**أحد** المكونات المهمة في الكثير من المتحكمات هو المحول التماثلي الرقمي ADC، وسيتم التركيز في هذا الفصل أساساً على كيفية استخدام المحول التماثلي الرقمي ADC الموجود داخل المتحكم atmega328 مع التطبيق على ذلك. ولكن، قبل أن ندخل في ذلك سنسوق بعض المفاهيم المهمة عن المحول الرقمي التماثلي دون الدخول في تفاصيل الطرق المختلفة لعملية تحويل أي إشارة من الصورة التماثلية إلى الصورة الرقمية لأنه توجد الكثير من المراجع التي تشرح ذلك بالتفصيل. على فكرة، تذكر جيداً أن تفاصيل الطرق المختلفة لهذا التحويل لن تكون ضرورية على الإطلاق لما سيقدم في هذا الفصل إذا فهمنا المفاهيم البسيطة التي سنقدمها هنا، ولكن معرفتها لن تضر ولكنها بالتأكيد ستكون مفيدة.

إننا نعيش الآن في عالم مليء بالإشارات التي نتعامل معها في كافة نواحي حياتنا اليومية بدأً من الأجهزة المنزلية إلى الموبايل والإنترنت وغيرها. هذه الإشارات توجد في واحدة من صورتين، الأولى هي الصورة التماثلية، والثانية هي الصورة الرقمية. الصورة التماثلية للإشارات مثل جهد القدرة الذي نتعامل معه ٢٠ فولت، ٥٠ هرتز، وخرج معظم الحساسات مثل حساسات الحرارة والضغط، وإشارة الصوت، وغيرها الكثير. هذا النوع من الإشارات بالرغم من كثرة وجوده حولنا إلا أن التعامل معه بمنتهى الصورة أصبح قليلاً هذه الأيام حيث يتم تحويله إلى الصورة الثانية وهي الصورة الرقمية قبل أن يتم التعامل معه، وذلك لعدة مميزات ستأتي ذكرها في معرض الحديث هنا.

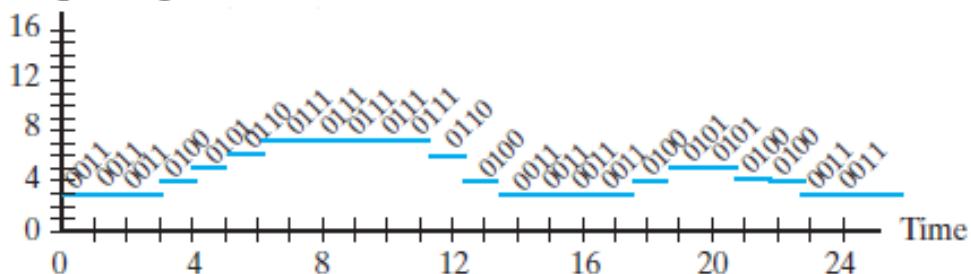
### الإشارات التماثلية والإشارات الرقمية



شكل ١-٧ الإشارات التماثلية وتحويلها إلى إشارات رقمية

شكل ١-٧ يبين ثلات أشكال للإشارات. الإشارة التماثلية أو المستمرة أو الانسياقية وهي كما نرى في شكل ١-٧ تأخذ مالانهاية من القيم فيما بين قيمتها الصغرى والعظمى. فمثلاً إذا كانت لدينا إشارة صوت خارجة من الميكروفون، أو إشارة حرارة خارجة من حساس لدرجة الحرارة، وتم تكبيرها ليحصر مقدارها بين الصفر و ٥ فولت، فإن هذه الإشارة يمكنها أن تأخذ أي قيمة بين القيمتين الصغرى والعظمى، مثل ١.٣٤٨٥٤ فولت و ٠.٠٠٠٠٠٥ وهكذا. هذه الصورة للإشارة لا يمكن التعامل معها من خلال الحاسوب أو المتحكم أو المعالج لأن كل منهم لا يتعامل إلا مع الإشارات الرقمية، ولذلك لابد من تحويل هذه الإشارات التماثلية إلى الصورة الرقمية. أول خطوات هذا التحويل هو أخذ عينات من الإشارة على فترات زمنية محددة ومنتظمة كما في شكل ١-٧ بـ. مقدار كل عينة من هذه العينات يكون غير محدد، بمعنى أن هذا المقدار يمكن أن يأخذ مالاً نهاية من القيم أيضاً بين القيمتين الصغرى والكبرى للإشارة. لذلك فإن الخطوة التالية من عملية تحويل الإشارة التماثلية إلى رقمية هي تحديد مقادير عينة لكل واحدة من هذه العينات. تتم هذه الخطوة عن طريق تحديد مدى القيم ما بين القيمة الصغرى والعظمى إلى عدد محدد من المستويات كما في شكل ١-٧ تـ، ويتم تقريب قيمة كل عينة من العينات إلى أقرب مستوى من هذه المستويات. على سبيل المثال، إذا فرضنا استخدام ١٦ مستوى لتمثيل كل مقادير هذه العينات، فإن كل عينة سيتم تمثيلها برقم من ٤ بتات  $(2^4 = 16)$  بدءاً من الرقم ٠٠٠٠ الذي يقابل أقل قيمة، حتى الرقم ١١١١ الذي يمثل أكبر قيمة ممكنة. لذلك، فإنه في هذه الحالة ستصبح الإشارة عبارة عن مجموعة من الأرقام التي تتراوح ما بين الرقم ٠٠٠٠ حتى الرقم ١١١١ وعلى فترات زمنية محددة وهي فترات أخذ العينات. إذن باختصار، فإن **التحول التماثلي الرقمي ADC** يقوم بأخذ عينات من الإشارة التماثلية على فترات زمنية محددة ثم يقرب قيمة كل عينة إلى أقرب مستوى من مستويات التقسيم الممكنة وتعرّيف العينة برقم هذا المستوى، وبذلك تصبح الإشارة عبارة عن أرقام ثنائية متواالية زمنياً كما في شكل ٢-٧ الذي يوضح إشارة الجهد المناسب مع درجة الحرارة (خرج الحساس) بعد تحويلها إلى الصورة الرقمية من ٤ بتات.

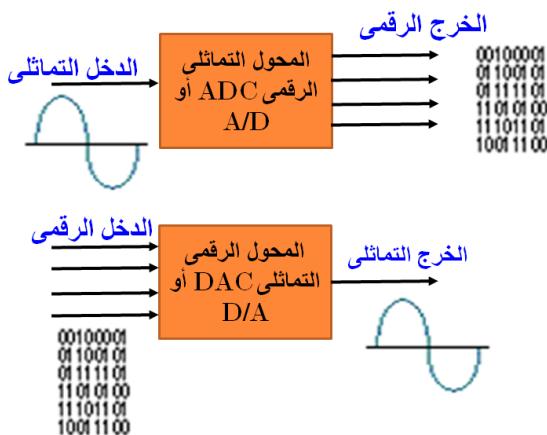
### المستويات الرقمية



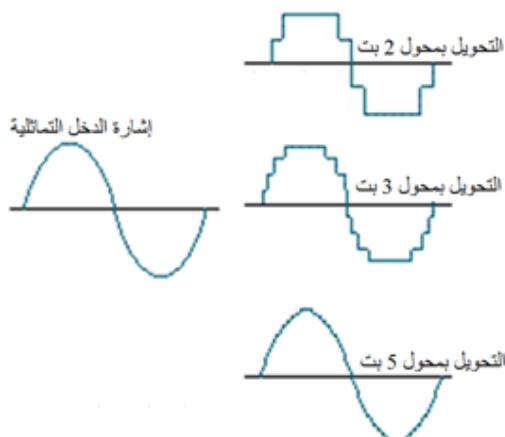
شكل ٢-٧ تمثيل مقدار درجة الحرارة في ١٦ مستوى (٤ بتات)

شكل ٣-٧ يبين رسم صندوقيا للمحول التماثلي الرقمي ADC أو A/D، والمحول الرقمي التماثلي DAC أو A/D، حيث يقوم بتحويل الإشارات الرقمية إلى تماثلية عند الحاجة لذلك.

كما رأينا فإن عدد بتات خرج المحول التماثلي الرقمي يحدد عدد مستويات تقسيم الإشارة التماثلية، فإذا كان عدد البتات ٤ مثلاً، فإن عدد مستويات التقسيم سيكون ١٦ مستوى منها المستوى صفر، وإذا كان ٨ بتات، فإن عدد مستويات التقسيم يكون ٢٥٦، أي ٨، أي ٢٥٦ مستوى، وهكذا. المحولات التماثلية الرقمية الشائعة تعطي خرجها مثلاً في ٨ بت، وهناك القليل منها يعطي الخرج في ١٠ بت، والقليل جداً منها يعطي الخرج في ١٢ بت، والنادر منها يعطي الخرج في ١٦ بت. مما هو تأثير عدد بتات الخرج على أداء المحول التماثلي الرقمي.



شكل ٣-٧ رسم صندوقى للمحول التماثلي الرقمي  
والمحول الرقمي التماثلي DAC



شكل ٤-٧ تأثير زيادة عدد بتات المحول على جودة عملية تحويل الإشارة من الصورة التماثلية إلى الصورة الرقمية

كانت مقدرتها على تمييز أو تحديد التغيرات في مقدار إشارة الدخل التماثلية أفضل، وهذا هو ما يسمى التحديدية أو

## التحديدية resolution

إن عدد بتات الخرج يعطى عدد مستويات تقسيم الإشارة التماثلية حيث يكون عدد هذه المستويات يساوى  $(2^n)$  عدد بتات الخرج (١)، فمثلاً إذا كان عدد بتات الخرج يساوى ٨، فإن عدد مستويات تقسيم الإشارة سيكون ٢٥٥ مستوى. إذا افترضنا أن أكبر قيمة لجهد إشارة الدخل تساوى ٥ فولت، فإن ذلك يعني أن هذه الخمسة فولت سيتم تقسيمتها إلى ٢٥٥ جزء، وبالتالي سيكون كل قسم يساوى ٢٠ ميللى فولت تقريباً. معنى ذلك أن أقل مقدار جهد للدخل يمكن أن يشعر به المحول سيكون ٢٠ ميللى فولت، وكلما كان هذا الجهد أقل كلما كان المحول أفضل وأغلى سعراً، لذلك كلما زاد عدد بتات المحول كلما

يمكن أن نقول عليها التمييزية، أو المقدرة التحليلية أو resolution. إذن يمكننا أن نعرف التحديدية للمحول التماثلي الرقمي ADC بأنها قدرة المحول على التمييز أو التفريق بين تغيرات الإشارة الرقمية وهي تساوى  $(1 - 2^N)/1$  حيث N هي عدد برات المحول. أحياناً تعطى التحديدية كنسبة مئوية بضرب المعادلة السابقة في مائة. لذلك فإن عدد برات خرج المحول يكون عاملاً مؤثراً جداً في سعر المحول التماثلي الرقمي. شكل ٤-٧ يبين تأثير زيادة عدد برات الخرج على جودة عملية التحويل.

### سرعة التحويل

من الخواص أو المعاملات المهمة لأى محول تماثلى رقمى هى سرعة تحويله لأى عينة من الصورة التماثلية إلى الصورة الرقمية. تتوقف هذه السرعة على الطريقة التي يتبعها المحول في تحويل العينات التماثلية من الإشارة إلى صورتها الرقمية. هناك العديد من الطرق المختلفة لهذا التحويل التي تختلف فيما بينها في سرعة التحويل، وهذا الكتاب ليس مجالاً لشرح أو تقديم هذه الطرق المختلفة. بعض هذه الطرق تتميز بالسرعة العالية، أى أن زمن التحويل فيها يكاد يكون صفر من الثواني، والبعض الآخر تكون سرعته قليلة بحيث أنه لا يناسب الكثير من التطبيقات العملية. يتوقف سعر المحول بدرجة كبيرة على سرعة تحويلة. الطريقة الأكثر شيوعاً في التحويل من الصورة التماثلية إلى الرقمية هي طريقة التقريب المتتالي successive approximation والتي تأخذ حوالي ٢٠ ميكروثانية في عملية التحويل، وهذه السرعة مناسبة للكثير من التطبيقات العملية، وهي الطريقة المستخدمة في الكثير من المحولات الموجودة داخل المتحكمات ومنها المتحكم .atmega328

### المعالجة في الزمن الحقيقي real time processing

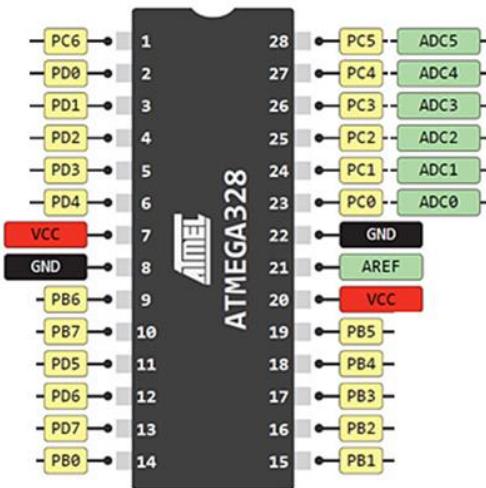
من القواعد المهمة التي يجب تذكرها جيداً لتحديد سرعة المحول التماثلي الرقمي الذي سيتم استخدامه في أي تطبيق هي أنه لكي يمكن استرجاع كل المعلومات التي في أي إشارة تماثلية عند تحويلها إلى الصورة الرقمية ثم إعادة تحويلها إلى الصورة التماثلية مرة أخرى أن يكون معدل أخذ العينات من الإشارة التماثلية sampling rate ضعف أكبر تردد في الإشارة التماثلية على الأقل، ولنضرب مثالاً على ذلك. إفترض أن لدينا إشارة صوت خارجة من ميكروفون، وتم تكبيرها وأخذ عينات منها وإدخالها على محول تماثلي رقمي لتحويلها إلى الصورة الرقمية لتخزينها في ذاكرة الحاسب، وبعد تسجيلها نريد استرجاعها من هذه الذاكرة وتحويلها مرة ثانية إلى الصورة التماثلية قبل إدخالها على سماعة من أجل أن نسمعها. الإشارة الصوتية بما فيها الموسيقى يمتد تردداتها حتى ٢٠ كيلوهertz. لكي تأخذ عينات من هذه الإشارة لكي

يتم إدخالها على المحول التماثلي الرقمي، فإن تردد أخذ العينات يجب أن يكون ٤ كيلوهرتز على الأقل، حتى يمكن سماع إشارة الصوت بجودة معقولة ويمكنك تمييز المتكلمين من خلال هذه الإشارات. إذا قل معدل أخذ العينات عن الأربعين كيلوهرتز، ستكون إشارة الصوت غير مرضية، وربما لن تستطع تمييز المتكلمين إذا بعثت كثيراً عن هذا التردد. باعتبار أن معدل أخذ العينات سيكون ٤ كيلوهرتز، فإن الزمن بين كل عينتين من الإشارة سيكون مقلوب هذا الرقم وهو ما يساوى ٢٥ ميكروثانية. في أثناء هذا الزمن بين كل عينتين (٢٥ ميكروثانية) يجب أن يتم أخذ العينة، وتحويلها إلى الصورة الرقمية، وتخزينها في الذاكرة، بالإضافة إلى أي زمن آخر قد يتم استخدامه في معالجات أخرى للإشارة. هذه الأزمنة كلها يجب أن تكون أقل من ٢٥ ميكروثانية، وإلا فإن أداء عملية التحويل لن يكون مرضياً للمستخدم، ومن هنا تتحدد سرعة المحول المراد استخدامه.

كما سنرى بعد قليل فإن المحول الرقمي التماثلي في المتحكم atmega328 يجب ألا يزيد تردد أخذ العينات المستخدم فيه عن ٢٠٠ كيلوهرتز (من مواصفات المتحكم)، وحيث أن هذا المحول يستخدم طريقة التقريب المتتالي، وهذه الطريقة تستغرق زمن مقداره يساوى عدد بتات المحول في أي عملية تحويل، أي أن هذا المحول سيستغرق زمن مقداره ١٠ نبضات تزامن لأن عدد بتاته يساوى عشرة (أيضاً من مواصفات المتحكم). إذن معنى ذلك أن معدل أخذ العينات بهذا المحول سيكون ٢٠ كيلوهرتز على الأكثـر. معنى ذلك أن تردد الإشارة التماثـلية التي سيتم تحويلها عن طريق هذا المحول يجب ألا يزيد عن نصف هذا التردد، أي ١٠ كيلوهرتز. ولذلك فإن هذا المحول مناسب لتحويل الإشارات الصوتية، أي الصوت الأدمي فقط بدون الموسيقى، حيث أن الصوت الأدمي يبلغ أقصى تردد له ٣,٥ كيلوهرتز تقريباً. إلى هنا سنكتفى بهذا القدر عن المفاهيم الأساسية لعملية التحويل التماثلي إلى الرقمي والتي كان من المهم ذكرها هنا حتى يستطيع القارئ الذي ليس لديه فكرة عن هذا النوع من المحولات والغرض من استخدامها، أن يستمر معنا وهو على دراية بكل المصطلحات التي سترد في معرض هذا الكلام. من يريد أي معلومات زيادة يمكنه اللجوء إلى أي من المراجع في ذلك أو الإنترنت التي يمكنه أن يجد عليها الكثير من الدروس بكل اللغات في هذا الشأن.

## ٢-٧ خصائص المحول التماثلي الرقمي في المتحكم atmega328

المحول التماثلي الرقمي في المتحكم atmega328 له خرج يتكون من ١٠ برات، مما يعني أن تحديدية هذا المحول أو مقداره التحليلية تساوى ١/١٠٢٣، وهذه تحديدية ممتازة للكثير من التطبيقات. هذا المحول يأخذ دخله من خلال متعدد مداخل أو متنقى وهذا المتنقى له سبعة مداخل تماثلية، ستة منها على أطراف البوابة C كما سنرى بعد قليل، يتم اختيار أحد هذه المداخل وتوصيله على دخل المحول ليقوم



شكل ٥-٧ أطراف المحول atmega328  
المستخدمة مع المحول التماثلي الرقمي ADC

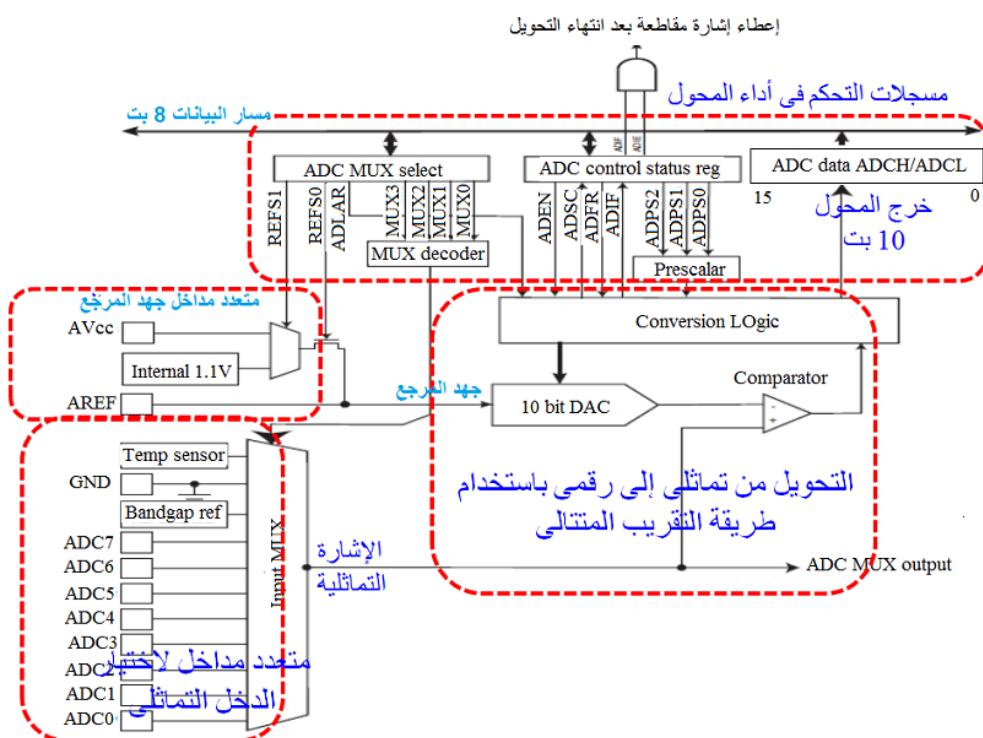
تحتاج للجهد VCC والأرضي كطرف قدرة، ويجب أن يكون الجهد VCC خالى من الضوضاء بقدر الإمكان، وذلك بتوصيله على مكثف تتعيم.

يمكن استخدام المحول بأحد طريقتين كما سنرى، إما طريقة التحويل الأحادي، أو طريقة التحويل المستمر. في طريقة التحويل الأحادي، يقوم المستخدم بإعطاء نبضة البدأ في التحويل، حيث يقوم المحول بعدها بإتمام عملية تحويل واحدة، ويتوقف، إلى أن يتم إعطاؤه نبضة تحويل أخرى. في الطريقة الثانية وهي طريقة التحويل المستمر، يقوم المستخدم بإعطاء نبضة واحدة للبدأ في التحويل، حيث يقوم بعدها المحول بإتمام عملية التحويل، وبعد إتمام هذه العملية يبدأ بنفسه في إجراء عمليات التحويل الأخرى آلياً وباستمرار إلى أن يتم إيقافه.

يمكن التعامل مع العشرة برات الكاملة من خرج المحول، وفي هذه الحالة فإن النتيجة ستوضع في مسجلين أحدهما يحتوى الثمانية برات العظمى من النتيجة ومسجل آخر يحتوى أول اثنين برت، وهذه الطريقة بالطبع ستحتاج لوقت أكثر في

التعامل بسبب أن مسجلات المتحكم كلها مسجلات 8 بت بينما نتيجة التحويل تتكون من 10 بت. يمكن تشغيل المحول بطريقة أخرى يتم فيها استخدام آخر 8 بت فقط من نتيجة التحويل وإهمال أول اثنين بت، وهذه الطريقة يفضلها الكثير من المبرمجين حيث أنها ستوفر وقت، كما أن أول اثنين بت من العشرة يمثلان في الغالب نسبة صغيرة جداً من القيمة الحقيقية للمتغير الذي يتم تحويله، بحيث يمكن اعتبارها كضوضاء لن يكون هناك خسارة كبيرة من إهمالها.

يبين شكل ٦-٧ نظرة شاملة على كل العناصر التي يتكون منها المحول التماثلي الرقمي داخل المتحكم atmega328. العناصر الأساسية في هذا الشكل هي: **المحول التماثلي الرقمي بطريقة التقرير المتتالي successive approximation** التي تتكون بدورها من محول تماثلي رقم DAC، ومقارن، ودائرة منطقية خاصة بهذه الطريقة. الدخل لهذا الجزء هو الإشارة التماثلية القادمة من **متعدد مداخل اختيار إشارة الدخل**. هذا المتعدد له ١٦ مدخلًا منها الستة الخاصة بالبوابة C والتي ستتكلم عنها بالتفصيل فيما بعد. هذا المتعدد سيحتاج لأربع بتات يتم اختيار أحد هذه المداخل عن طريق الشفرة التي سيتم وضعها على هذه البتات، وهذه البتات قادمة من الجزء العلوي الذي يحتوى مسجلات التحكم في أداء المحول والتي سيتم شرحها بالتفصيل.

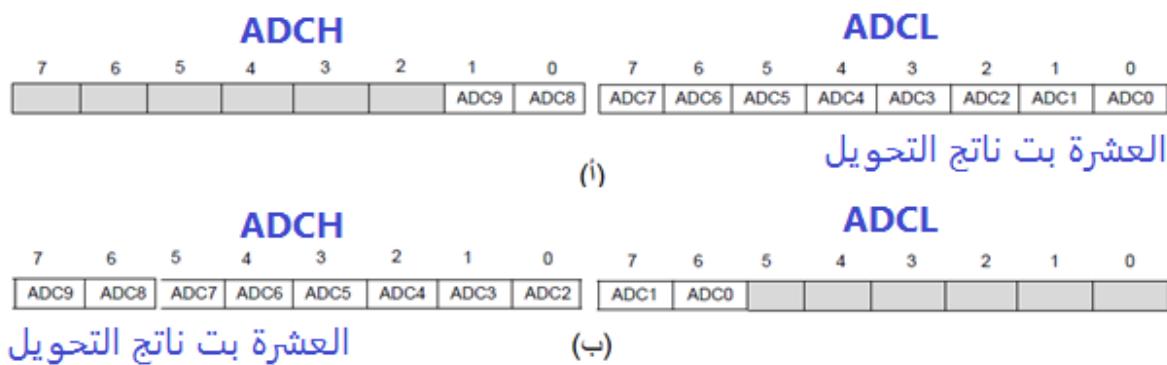


شكل ٦-٧ نظرة شاملة على عناصر المحول التماثلي الرقمي داخل المتحكم atmega328

هناك أيضاً متعدد مداخل خاص باختيار جهد المرجع اللازم لطريقة التقرير المتتالي، وسيتم الحديث عليه بالتفصيل أيضاً. أخيراً هناك الجزء الخاص بمسجلات التحكم في أداء المتحكم وهي في الأساس ثلاثة من مسجلات سيأتي شرحها بالتفصيل أيضاً.

### ٣-٧ مسجل بيانات المحول التماثلي الرقمي

بعد أن يستكمل المحول عملية التحويل يضع نتيجة التحويل في المسجلين ADCH, ADCL حيث أن النتيجة تكون ممثلة في ١٠ برات. يعتبر هذين المسجلين مسجل واحد من ١٦ بت بحيث يكون المسجل ADCL هو المسجل ذو القيمة الأقل، والمسجل ADCH هو المسجل ذو القيمة الأعلى. هناك طريقتان يتم بهما وضع النتيجة المكونة من ١٠ برات في هذين المسجلين، ويتم ذلك عن طريق البت ADLAR وهي البت رقم ٥ في مسجل الاختيار من متعددات المدخلات ADMUX الذي سيأتي شرحة في الجزء التالي. عند وضع البت  $ADLAR = 0$  وهي القيمة التلقائية لها يتم محاداة العشرة برات ناحية اليمين في المسجلين ADCH, ADCL بحيث أن أول ثمان برات من النتيجة توضع في المسجل ADCL، وآخر اثنين بت من النتيجة توضعان في أول اثنين بت من المسجل ADCH كما في شكل ٧-٧أ. عند وضع البت  $ADLAR = 1$ ، تحدث محاداة للنتيجة ناحية اليسار، بحيث أن أول اثنين بت من النتيجة التحويل توضعان في آخر اثنين بت من المسجل ADCL، وآخر ثمان برات من النتيجة توضع في المسجل ADCH بالكامل كما في شكل ٧-٧ب.



شكل ٧-٧ محاداة ناتج التحويل (العشرة برات) ناحية اليمين أو اليسار في المسجلين ADCH و ADCL

عند قراءة نتيجة التحويل من المسجلين ADCL و ADCH، فإنه يجب قراءة ADCL أولاً، حيث عندها لن يتم تحديد بيانات التحويل حتى يتم قراءة المسجل ADCH حتى تكون قراءة المسجلين متزامنة. أي أن بيانات التحويل لا يتم تحديدها دائماً إلا بعد قراءة المسجل ADCH. في حالة محاذاة البيانات ناحية اليسار، فإنه يمكن في هذه الحالة قراءة المسجل ADCH فقط حيث ستكون هي البيانات الثمانية العليا في النتيجة، وفي هذه الحالة يمكن الاكتفاء بقراءة المسجل ADCH فقط وإهمال الاثنين بت التي في المسجل ADCL واعتبارهما ضوضاء كما ذكرنا، وفي هذه الحالة ستكون عملية قراءة نتيجة التحويل بسيطة جداً، حيث ستقتصر على قراءة المسجل ADCH فقط. نتيجة التحويل الرقمية يمكن حسابها من المعادلة  $ADC = \frac{1024}{V_{ref}} V_{in}$ ، حيث  $V_{in}$  هو قيمة الدخل التماثلي، و  $V_{ref}$  هو الجهد المرجعي المختار. عندما  $V_{in}=V_{ref}$  فإن  $ADC=1024$  أي عشرة وحدات في النظام الثنائي.

#### ٤-٧ مسجل الاختيار من متعددات المدخل

#### ADC Multiplexer Selection Register, ADCMUX

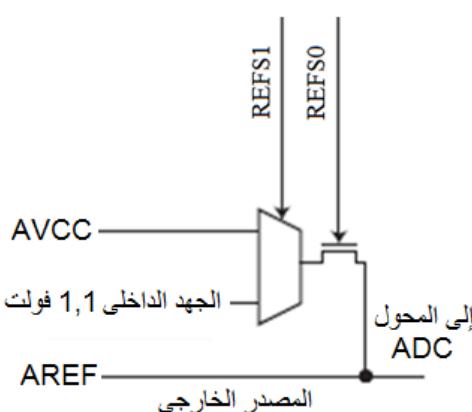
7	6		5	4		3	2	1	0
REFS1	REFS0		ADLAR			MUX3	MUX2	MUX1	MUX0
R/W	R/W		R/W			R/W	R/W	R/W	R/W
0	0		0			0	0	0	0

شكل ٨-٧ مسجل الاختيار من متعددات المدخل ADCMUX

هذا المسجل يحتوى أربع باتات يتم عن طريقها اختيار الدخل التماثلى الذى سيتم تحويله من خلال متعدد مداخل الدخل التماثلى، واثنين بت لاختيار مصدر الجهد الذى سيستخدم كجهد مرجع في طريقة التحويل الممتالى، وبت خاصة بمحاذاة نتيجة التحويل يميناً أو يساراً كما شرحنا مسبقاً، وبت غير مستخدمة، وكل ذلك موضح في شكل ٧-٨، وسنوضح تفاصيل ذلك فيما يلى:

**الآيات ٦ و ٧، بثات اختيار الجهد المراجع** **REFSn**

تستخدم هاتين البتين في اختيار الجهد المرجعى للمحول التماثلى الرقمى، والجدول ١-٧ يبين جميع الاختيارات الممكنة منها. إذا تم تغيير هاتين البتين أثناء أى عملية تحويل، فإن تأثير ذلك لن يؤخذ فى الاعتبار إلا بعد الانتهاء من عملية التحويل، الجارية. شكل ١-٧ يبين تفاصيل عمـا هذه الدائرة، التي تتكون من متعدد مداخل له دخـلـين فقط (AVCC)،



## شكل ٩ اختيار الجهد المرجعي

الجهد الداخلى ١٥ فولت)، ويتم التحكم فيه بالبت REFS1 التي عندما تكون صفر يتم اختيار الدخل AVCC وعندما تكون واحد يتم اختيار الجهد الداخلى ١٥ فولت. خرج هذا المتعدد، REFS0، موصى على ترانزistor يتم التحكم فى تشغيله بالبت REFS0 بحيث عندما تكون هذه البت صفر فإن الترانزistor يكون مفتوحا، open circuit، وعندما تكون هذه البت تساوى واحد يكون الترانزistor short circuit. عندما تكون البت REFS0 بصفر، يكون الترانزistor مفتوحا، وفي هذه الحالة يمكن توصيل الجهد المرجعى الخارجى AREF الذى يتم توصيله على الطرف ٢١ لشريحة المتحكم. حاول قراءة صفوف الجدول ١-٧ ومطابقتها بالل

## جدول ١-٧ بات اختيار الجهد المرجعي

REFS1	REFSO	الجهد المرجعى المختار
0	0	إغلاق المصادر الداخلية واختيار AREF الجهد المرجعى الخارجى
0	1	اختيار الجهد AVCC داخلياً، مع وضع مكثف تتعيم على الطرف خارجياً وتوصيله بالأرضي AREF
1	0	غير مستخدم
1	1	اختيار الجهد الداخلى ١,١ فولت، مع توصيل مكثف تتعيم أيضاً على الطرف خارجياً وتوصيله AREF بالأرضي

هذه البت كما شرحناها مسبقاً عندما تكون صفر  
ستجعل نتيجة التحويل محاذاة ناحية اليمين في المسجلين  
ADCH و ADCL وهذا هو الوضع التقليدي، وعندما  
تكون بوحدة واحدة ستجعل نتيجة التحويل محاذاة ناحية اليسار.  
تغيير البت ADLAR سيؤثر فوراً في محاذاة نتيجة  
التحويل الموجودة في مسجل البيانات ADCL و ADCH.

البت ٤: غير مستخدمة

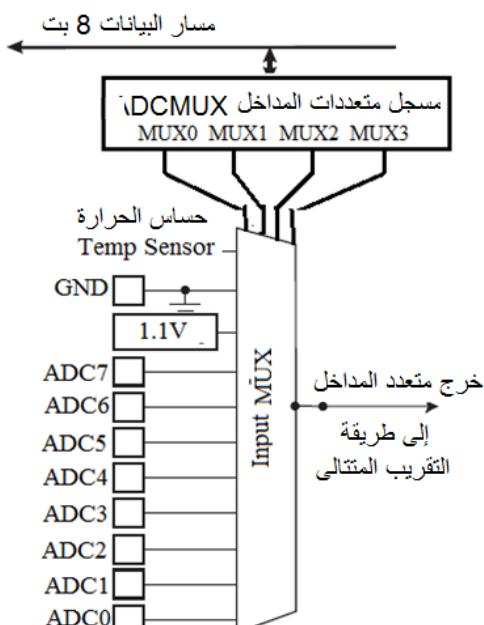
## MUX3:0 و 2 و 3 باتات اختيار إشارة الدخل التماثلية

هذه الباتات الأربع يتم بها اختيار واحد من ستة عشر مصدراً للإشارات التماثلية. هذه المصادر والشفرة المقابلة على هذه الباتات موضحة في جدول ٢-٧، كما إن شكل ١٠-٧ يبين رسمياً تخطيطياً لهذا المتعدد. المصادر السبعة الخاصة بالبوابة C، والشفرات المقابلة لها هي ٠٠٠٠ حتى ٠١١١، وكما نعلم فإن البوابة C لها سبعة أطراف فقط في هذا المحكم. الجدير بالذكر أيضاً أن الدخل PC6 أيضاً من البوابة C غير مستخدم كدخل تماثلي في شرائح المحكم ذات الصفين من الأرجل الموضحة في شكل ٥-٧. الشفرة الثامنة وهي ١٠٠٠ تقوم بإدخال خرج حساس الحرارة الموجود داخل المحكم ليتم تحويله إلى الصورة الرقمية. الشفرات من تسع، ١٠٠١، حتى ١٣، ١١٠١ غير مستخدمة أو ممحوزة للاستخدام المستقبلي كما يقول دليل شريحة المحكم. الشفرة ١١١٠ تقوم بإدخال الجهد الثابت ١,١ فولت من داخل الشريحة ليتم تحويله إلى الصورة الرقمية، وأما الشفرة الأخيرة لهذه الباتات، ١١١١، فتقوم بإدخال الأرضي (zero volt)، إلى الحول ليتم تحويله إلى الصورة الرقمية، والمهدف من ذلك هو التأكد من التشغيل الصحيح للمحول التماثلي الرقمي، حيث من المفترض في هذه الحالة أن يعطي خرجاً كله أصفار ٠٠٠٠٠٠٠٠.

جدول ٢-٧ المداخل المختلفة لمتعدد

مداخل الإشارات التماثلية

MUX[3:0]	المدخل التماثلية
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	حساس درجة الحرارة
1001	غير مستخدم
1010	غير مستخدم
1011	غير مستخدم
1100	غير مستخدم
1101	غير مستخدم
1110	١.١V ( $V_{BG}$ )
1111	٠V (GND)



شكل ١٠-٧ متعدد مدخلات الإشارات

## قياس درجة الحرارة

يعتمد قياس درجة الحرارة على حساس لدرجة الحرارة موجود داخل المتحكم ويتم تنشيطه بتوصيل خرجه على دخل المحول التماثلي الرقمي بوضع الشفرة 1000 على خطوط الاختيار لمتعدد المدخل MUX3:0. يجب أيضا في هذه الحالة اختيار الجهد المرجعى ١,١ فولت من خلال متعدد مداخل الجهد المرجعى الذى سبق شرحه. يمكن قراءة جهد الخرج لحساس الحرارة عن طريق أخذ عينة من هذا الجهد ممرة واحد كلما دعت الحاجة لذلك. جهد الخرج من حساس

جدول ٣-٧ تغير جهد خرج

حساس درجة الحرارة

جهد الحرارة	جهد الخرج
٢٤٢ ميلي فولت	٠٤٥-
٣١٤ ميلي فولت	٠٢٥+
٣٨٠ ميلي فولت	٠٨٥+

درجة الحرارة يتتناسب خطيا تقريبا مع درجة الحرارة حيث ترتفع درجة الحرارة بمقدار واحد ميلي فولت لكل ارتفاع في درجة الحرارة بمقدار درجة واحدة مئوية ( $1\text{mV}^{\circ}\text{C}$ ). جدول ٣-٧ يبين ثلاث قراءات لدرجات حرارة فعلية من خرج الحساس تعكس هذه العلاقة. نذكر القارئ بأنه قد يكون هناك اختلافا بسيطا في خرج حساس درجة الحرارة من شريحة متحكم لأخرى لذلك ينصح بعمل منحنى معايرة لكل شريحة متحكم على حده عند الرغبة في الحصول على دقة عالية لدرجة الحرارة.

## ٥-٧ مسجل التحكم والحالة A

### ADC Control and Status Register A, ADCSRA

هناك مسجلان للتحكم والحالة في المتحكم atmega328 أولهما هو المسجل A الذي سنشرح بقائه في هذا الجزء، وأما مسجل الحالة والتحكم B فسيأتي شرحه في الجزء التالي. شكل ١١-٧ يبين البنية الشمانية الموجودة في هذا المسجل، وأما وظيفة كل بت فسيتم شرحها فيما يلى:

7	6	5	4	3	2	1	0
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

شكل ١١-٧ مسجل التحكم والحالة A في المتحكم atmega328

**البت ٧: بت تنشيط المحول ADEN**

هذه البت خاصة بتنشيط المحول، حيث يوضع واحد في هذه البت يصبح المحول التماثلي الرقمي نشطاً أو مستعداً لعملية التحويل. وضع صفر في هذه البت سيُخمد المحول، حتى لو تم وضع هذه البت بصفر أثناء أي عملية تحويل فإن هذه العملية لن يتم استكمالها.

**جدول ٤-٧: معامل قسمة نبضات تزامن المحول التماثلي الرقمي**

ADPS[2:0]	معامل القسمة
000	2
001	2
010	4
011	8
100	16
101	32
110	64
111	128

**البت ٦: بدأ التحويل ADSC**

في طريقة التحويل الأحادية التي يتم فيها أخذ عينة واحدة فقط، فإنه لكي يتم هذه العملية لابد من أن تكون هذه البت تساوى واحد. في طريقة التحويل المستمر التي يتم فيها أخذ عينة وب مجرد الانتهاء من تحويل هذه العينة يبدأ المحول في أخذ عينة أخرى وباستمرار. لكي تبدأ هذه العملية لابد من أن تكون البت ADSC تساوى واحد مع أخذ أول عينة. أثناء عملية التحويل تظل هذه البت بواحد، وب مجرد الانتهاء من عملية التحويل تنزل هذه البت إلى الصفر تلقائياً.

**البت ٥: تنشيط القدح الآلي ADATE**

عند وضع هذه البت بواحد، يتم تنشيط القدح الآلي للمحول التماثلي الرقمي. مصدر هذا القدح الآلي يتم اختياره عن طريق وضع بات اختيار مصدر القدح الموجودة في مسجل التحكم والحالة B التي سيتم شرحها في الجزء التالي. عملية القدح تكون على الحافة الصاعدة لمصدر القدح.

**البت ٤: علم المقاطعة ADIF**

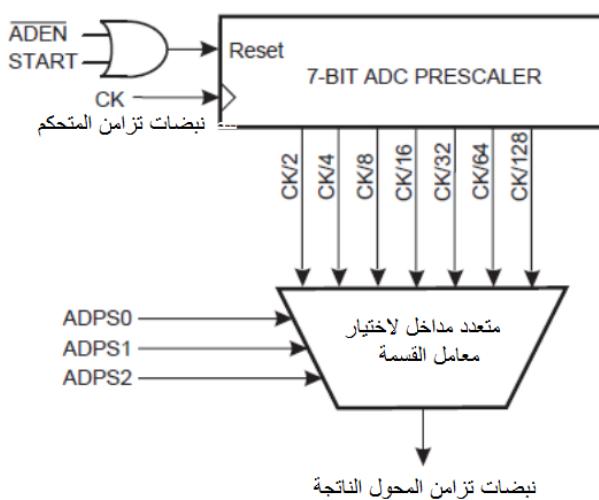
هذه البت عبارة عن علم يبين الانتهاء من عملية التحويل، حيث بمجرد الانتهاء من تحويل أي عينة وتحديد محتويات مسجلى بيانات الخرج ADCH و ADCL، فإن هذه البت تصبح بواحد. في هذه الحالة إذا كان قناع المقاطعة الخاص بالمحول التماثلي الرقمي ADIE قد تم تنشيطة، وكان علم المقاطعة العام I في مسجل الأعلام الخاص بالتحكم قد تم تنشيطة أيضاً، فإنه بمجرد أن تصبح هذه البت ADIF بواحد، فإن المتحكم سيقفز إلى برنامج خدمة المقاطعة

الخاص بهذه العملية. بمجرد الانتهاء من برنامج خدمة المقاطعة يتم تنزيل علم المقاطعة إلى الصفر تلقائيا، استعدادا لعملية التحويل التالية.

### البت ٣: قناع تنشيط المقاطعة ADIE

هذه البت تمثل قناع المقاطعة الخاص بالمحول التماثلي الرقمي. بوضع هذه البت تساوى واحد، فإنه إذا كان علم المقاطعة العام I يساوى واحد هو الآخر، فإنه بمجرد انتهاء أي عملية تحويل سيقفر المتحكم إلى برنامج خدمة المقاطعة كما أشرنا مسبقا.

### البّات ٢ و ١ و صفر، اختيار قاسم نبضات التزامن ADPSn



شكل ١٢-٧ اختيار نبضات تزامن المحول التماثلي الرقمي ADEN. بمجرد وضع الشفرة المناسبة لمعامل القسمة المطلوب، ثم تنشيط المحول التماثلي الرقمي بوضع البت Tzamn واحد، يبدأ المحول في العمل.

كما أشرنا مسبقا، فإن معظم طرق التحويل من تماثلى إلى رقمي ومنها طريقة التحويل المتتالي تحتاج إلى نبضات تزامن. هذه النبضات يتم أخذها من مصدر نبضات تزامن المتحكم وقسمتها على معامل معين بحيث تصبح هذه النبضات أقل من ٢٠٠ كيلوهertz وهي أقصى نبضات يمكن أن يعمل عندما المحول. جدول ٤-٧ يبين الشفرات التي يتم وضعها على هذه البّات الثلاثة ومعامل القسمة المقابل لكل حالة.

شكل ١٢-٧ يبين رسميا توضيحيا لقاسم نبضات

## ٦-٧ مسجل التحكم والحالة B

### ADC Control and Status Register B, ADCSRB

شكل ١٣-٧ يبين برات هذا المسجل حيث نلاحظ أن هناك ٤ برات من برات هذا المسجل غير مستخدمة وهي البار ٧ و ٥ و ٤ و ٣.

7	6	5	4	3	2	1	0
	ACME				ADTS2	ADTS1	ADTS0
R/W				R/W	R/W	R/W	R/W
0				0	0	0	0

شكل ١٣-٧ مسجل التحكم والحالة B في المتحكم atmega328

### البت ٦: تنشيط متعدد مداخل المقارن التماثلي Analog Comparator Multiplexer Enable, ACME

هذه البت عندما تكون بوحدة تنشط متعدد مداخل يتم به اختيار مصدر مقارنته باستخدام المقارن التماثلي، وسيأتي الشرح الكامل لهذا المقارن التماثلي فيما بعد.

### البار ٢ و ١ و صفر: برات اختيار مصدر القدح الآلي لحول المحوول الرقمي ADTS2:0

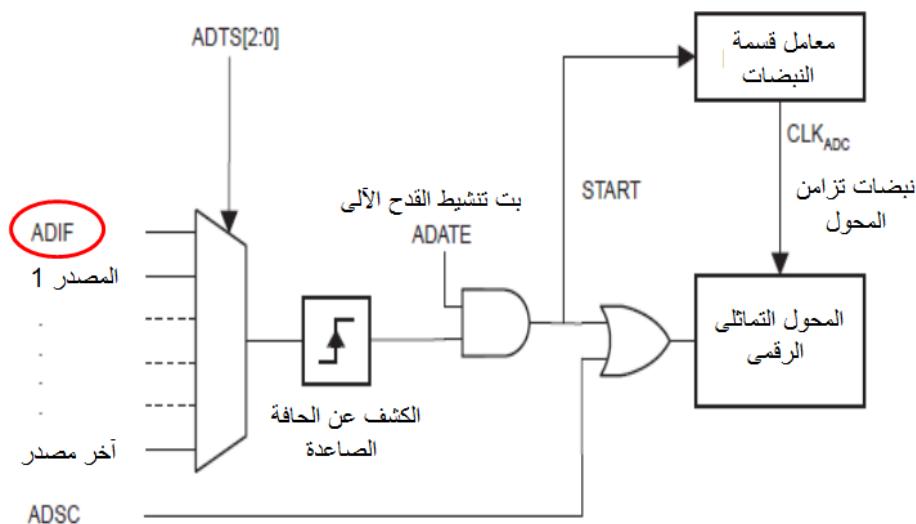
جدول ٧-٥ اختيار طريقة قدرح المحوول التماثلي الرقمي

ADTS[2:0]	Trigger Source	مصدر القدح
000	Free Running mode	طريقة التحويل الحر
001	Analog Comparator	المقارن التماثلي
010	External Interrupt Request 0	طلب المقاطعة الخارجية رقم صفر
011	Timer/Counter0 Compare Match A	المقارنة والتساوي من المؤقت رقم صفر
100	Timer/Counter0 Overflow	فيضان المؤقت رقم صفر
101	Timer/Counter1 Compare Match B	المقارنة والتساوي من المؤقت رقم ١
110	Timer/Counter1 Overflow	فيضان المؤقت رقم ١
111	Timer/Counter1 Capture Event	مسك حدث في المؤقت رقم ١

إذا كانت بـت تنشيط القدح الآلي للمحوول التماثلي الرقمي ADATE في المسجل السابق ADCSRA تساوى واحد، فإن البار ٢:٠ ADTS2:0 ستختار مصدر من ثمانية مصادر يتم عن طريقه القدح الآلي للمحوول التماثلي الرقمي.

القدح الآلى يقصد به أنه بمجرد انتهاء أي عملية تحويل تبدأ عملية تحويل جديدة آلياً. عملية التحويل الآلى المستمرة بالطبع تحتاج لمصدر لنبضات تمثل بدأ عملية التحويل في كل مرة، وهذه البتات كما ذكرنا تحدد هذه المصادر. ستبدأ عملية التحويل مع الحافة الصاعدة للمصدر المختار بهذه البتات. جدول ٥-٧ يبين هذه المصادر والشفرة المقابلة التي يتم وضعها في هذه البتات لاختيار هذه المصادر.

شكل ١٤-٧ يبين هذه المصادر وكيفية اختيارها من خلال متعدد مدخلات MUX خطوط الاختيار فيه هي البتات الثلاثة السابقة. كما نلاحظ فإن علم المقاطعة ADIF هو المصدر الأول وهو يمثل التشغيل الحر للمحول التماثلى الرقمى حيث بمجرد الانتهاء من أي عملية تحويل يتغير العلم ADIF من الصفر إلى الواحد، ومع هذه الحافة تبدأ عملية التحويل الجديدة وهو الاختيار الأول في جدول ٥-٧. باقى المصادر الموضحة في هذا الجدول وفي شكل ١٤-٧ تأتى من مصادر غير المحول التماثلى الرقمى مثل المؤقتات التي بداخل المتحكم نفسه، أو حتى من خارج المتحكم من خلال طرف المقاطعة INT0. نكرر هنا أنه لكي تبدأ عملية التحويل، فإنه لا بد من كتابة واحد في البت ADSC في المسجل ADCSRA حتى تبدأ عملية التحويل، ثم تستمر بعد ذلك بطريقة آلية تبعاً لنبضات هذه المصادر.



شكل ١٤-٧ اختيار مصادر القدح الآلى للمحول التماثلى الرقمى

## ٧-٧ أمثلة على تشغيل المحول التماشى الرقمي

سنقدم هنا بعض الأمثلة البسيطة التي نفهم منها كيفية تشغيل المحول التماشى الرقمي ADC من خلال محاكاة الدائرة المستخدمة على برنامج بروتس وكتابة البرنامج للمتحكم باستخدام برنامج الأتمل استديو.

### مثال ١

إدخال إشارة تماشية من خلال مقسم جهد (مقاومة متغيرة) على أحد مداخل البوابة C، ثم كتابة برنامج يقرأ القيمة الرقمية المنشورة للدخل التماشى بحيث إذا كانت القيمة الرقمية أكبر من ٥١٢ (نصف القيمة العظمى) يضيء الدايوه الضوئي الموصى على الطرف PD0، وإذا كانت القيمة الرقمية أقل من ٥١٢ يضيء الدايوه الضوئي على الطرف PD1. الدائرة المستخدمة في ذلك موضحة في شكل ١٥-٧، والبرنامج الذى سينفذ هذه المهمة سيكون كما يلى:

```
/*

```

```
* ADC1.c
```

```
*
```

```
* Created: 7/20/2017 9:10:31 AM
```

```
* Author : Mohamed eladawy
```

```
*/
```

```
#include <avr/io.h>
```

```
int main(void)
```

```
{
```

```
unsigned int adc_value; // متغير توضع فيه نتيجة التحويل
```

```
DDRD=0xff; // البوابة دى بوابة خرج
```

```
PORTD = 0x00; // أصفار على كل مخارج البوابة دى
```

```
ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS0);
```

```
// تنشيط المحول، ونسبة القسمة ٣٢
```

```
ADMUX=0x05; // اختيار الدخل من الطرف سى خمسة
```

```
while (1)
```

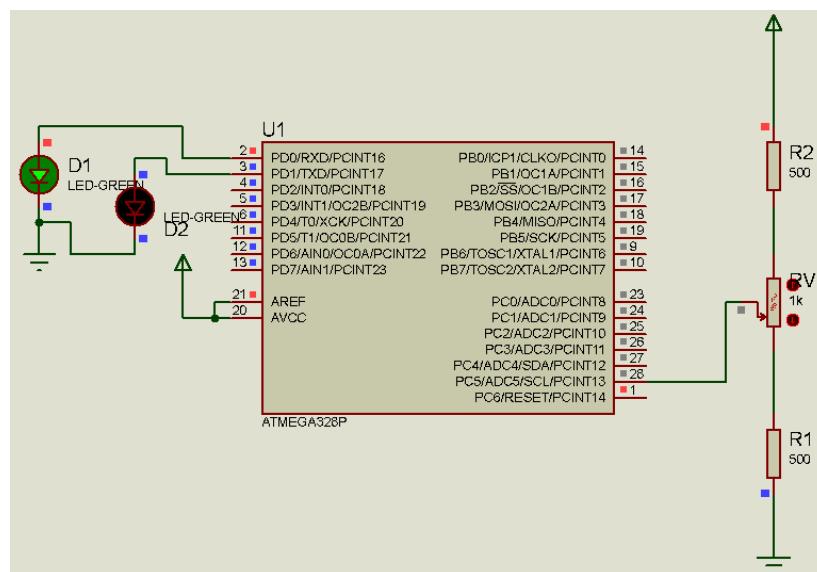
```
{
```

```
ADCSRA |= (1<<ADSC); // بدأ التحويل
```

```

while (ADCSRA & (1<<ADSC)); //انتظار حتى الانتهاء من التحويل
adc_value = ADCW; //قراءة نتيجة التحويل
if (adc_value<512)
    PORTD = 0x01; //إضاءة الدايدو الأول
else
    PORTD =0x02; //إضاءة الدايدو الثاني
}
}

```



شكل ١٥-٧ دائرة المثال ١

**مثال : ٢ :**

في هذا المثال سنستخدم الجهد التماثلي الخارج من مقسم الجهد كما في المثال السابق، ونخazi خرج المحول ناحية اليسار في المسجلين ADCH و ADCL و سنأخذ الثمانية بิตات العليا في المسجل ADCH فقط ونعرضها على ثمانية دايدو ضوئية كما في شكل ١٦-٧ والبرنامج الذى سيقوم بهذه المهمة سيكون كالتالى:

```

/*
* ADC2.c
*

```

\* Created: 7/21/2017 7:35:58 AM

\* Author : Mohamed eladawy

\*/

```
#include <avr/io.h>
```

```
int main(void)
```

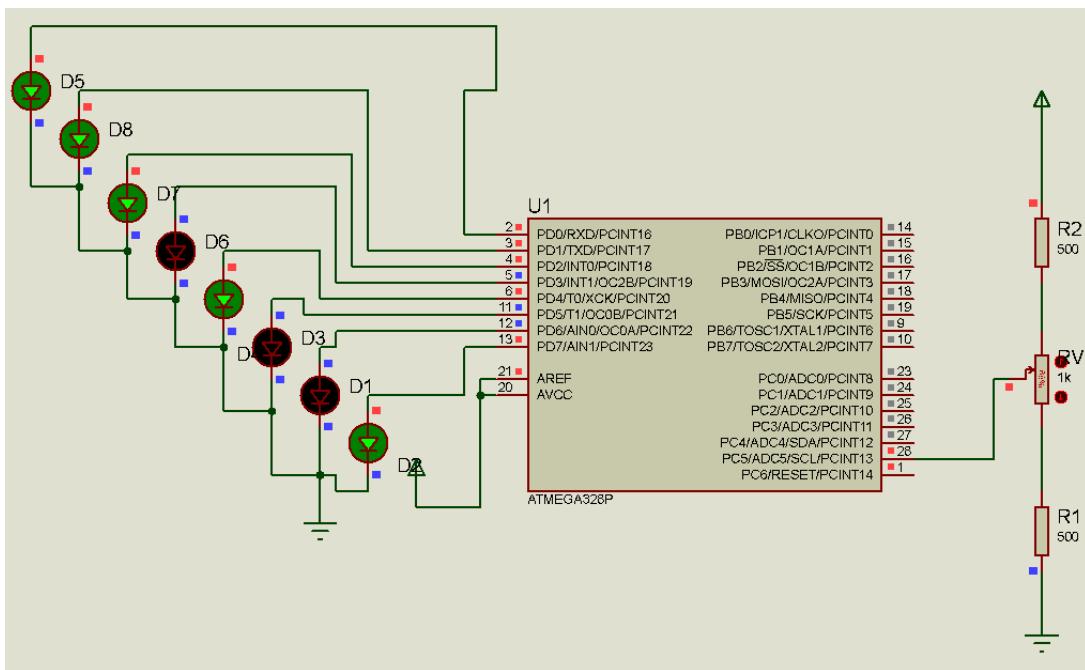
```
{
```

```
    DDRD=0xff;
```

```
    PORTD = 0x00;
```

```
    ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS0)|(1<<ADLAR);
```

```
    ADMUX=0x25;
```



شكل ١٦-٧ دائرة المثال ٢

```
while (1)
```

```
{
```

```
    ADCSRA |= (1<<ADSC);
```

```
    while (ADCSRA & (1<<ADSC));
```

```
    PORTD = ADCH;
```

```
{
}
```

## ٨-٧ مشروع تطبيقى على الخول التماشى الرقمي ADC

سنستخدم في هذا المشروع حساس لدرجة الحرارة وهو الحساس LM35 المعرف في برنامج بروتس والذي يعطى ١٠ ميللي فولت لكل درجة حرارة مئوية. ستدخل خرج حساس الحرارة على الدخل ٥ من البوابة C لكي يتم تحويله إلى الصورة الرقمية، ونعرض درجة الحرارة على شاشة LCD كما في شكل ١٧-٧. سنشغل الخول بطريقة القدح الأحادي، وعندما يتم الانتهاء من التحويل يتم مقاطعة المتحكم ويتم القفز إلى برنامج خدمة مقاطعة. لقد وضعنا في التمريرين ثلاث ملبات إضاءة يتم التحكم فيها في برنامج خدمة المقاطعة بحيث نضيء الأحمر إذا كانت درجة الحرارة أكبر من ٤٠ درجة مئوية، ونكتب على الشاشة عبارة "Very hot" في السطر الأول. عندما تكون درجة الحرارة بين ٣٥ و ٤٠ نضيء الضوئين الأحمر والأصفر ونكتب على الشاشة عبارة "Near hot". عندما تكون درجة الحرارة بين ٢٥ و ٣٥ نضيء اللمبة الخضراء فقط ونكتب عبارة "Very Nice Temp.". عندما تكون درجة الحرارة بين ٢٠ و ٢٥ نضيء الضوئين الأصفر والأخضر ونكتب عبارة "Near cold". عندما تكون درجة الحرارة أقل من ٢٠ درجة مئوية نضيء الضوء الأصفر فقط ونكتب عبارة "Very cold". بعد الانتهاء من برنامج خدمة المقاطعة يتم إعطاء نبضة بدأ تحويل جديد، وهكذا.

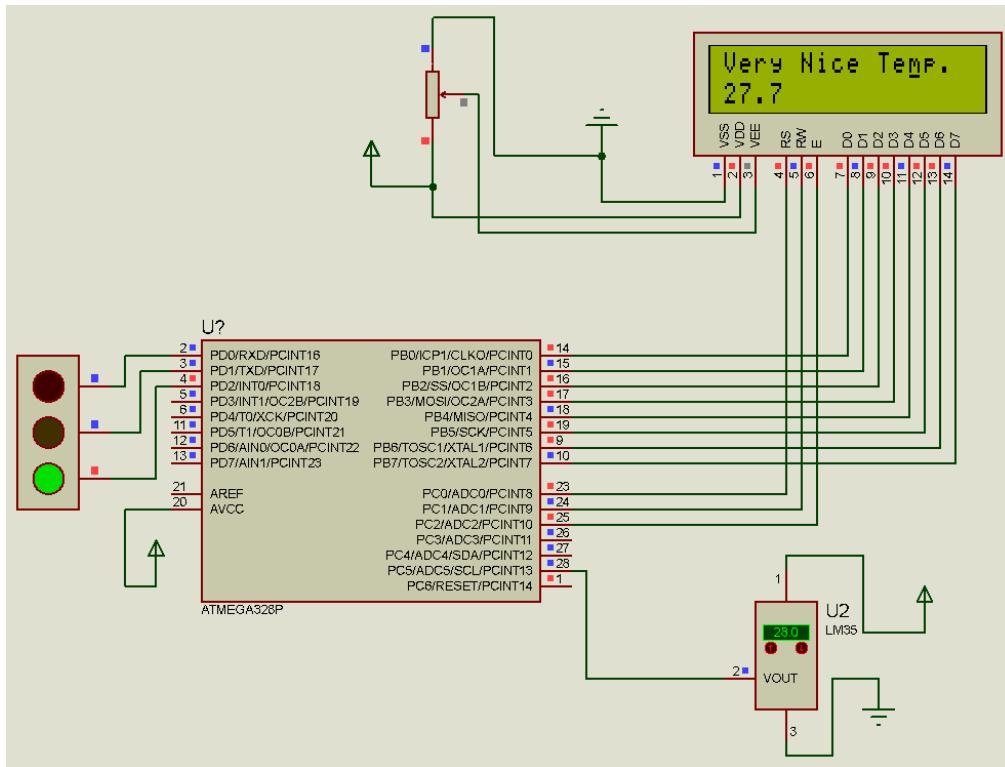
شكل ١٧-٧ يبين محاكاة هذا المشروع على البروتس، والبرنامج المستخدم سيكون كالتالي:

```
/*
* ADC4.c
*
* Created: 7/26/2017 12:44:26 PM
* Author : Mohamed Eladawy
*/
```

```
#include <stdlib.h>
#include <string.h>
#define F_CPU 1000000ul
#include <avr/io.h>
```

```
#include <avr/interrupt.h>
#include <util/delay.h>
#include "LCDlib.h"

volatile char tempC; // قيمة درجة الحرارة بعد تحويلها ستوضع في هذا المتغير
```



شكل ١٧-٧ مشروع عرض درجة الحرارة

```
int main(void)
{
    DDRC =0x0F;
    DDRD=0xFF;
    LCD_Init();
    ADCSRA |= (1<<ADEN) | (1<<ADPS2)
    |(1<<ADPS0)|(1<<ADATE)|(1<<ADIE
    ADMUX |= (1<<REFS0) | (1<<ADLAR) | (1<<MUX2)| (1<<MUX0);
```

مصدر القدرة المرجعى هو مصدر القدرة العام، وخرج الحساس على الدخل ٥ من البوابة سى، ومحاذاة النتيجة //  
ناحية اليسار

`sei();`

`while (1)`

```
{
}
```

`ADCSRA |= (1<<ADSC); // بدأ التحويل`

`while (ADCSRA & (1<<ADSC)); // انتظار انتهاء التحويل`

`ISR(ADC_vect)`

```
{
```

`char display[5]; // مصفوفة أحرف تحوى النتيجة لعرضها على الشاشة`

`tempC=ADCH*1.96;`

`if (tempC>40)`

```
{
```

`LCD_GoToLineOne();`

`LCD_DisplayString("Very Hot Te`

`itoa(tempC,display,10);`

`LCD_GoToLineTwo();`

`LCD_DisplayString(display);`

`LCD_DisplayString(".");`

`itoa(tempC%10,display,10);`

`LCD_DisplayString(display);`

`PORTD=0x01;`

```
}
```

`else if((tempC>35)&(tempC<40))`

تم محاذاة نتيجة التحويل ناحية اليسار واستخدمنا البایت العلیا فقط ADCH. حيث أن الجهد المرجعی تم اعتباره VCC، فإن تحديدية المحول ستكون  $\frac{5}{255} = 0.0196$  وهو ما يساوى ٢٠ ميللى فولت تقريباً لكل زيادة بمقدار واحد في خرج المحول. حيث أن الحساس يعطى ١٠ ميللى فولت لكل درجة حرارة، فإن درجة الحرارة المقابلة لقراءة المحول ستكون  $10 \times 0.0196 = 0.196$  درجة حرارة تقريباً، أي بضرب نتیجة التحويل في 1.96 تقريباً.

الدالة itoa(tempC,display,10) تقوم بتحويل المتغير tempC إلى سلسلة أحرف توضع في المتغير display تمهدًا لعرضها على الشاشة. الرقم 10 هو قاعدة الدالى يتبعها المتغير tempC وهي النظام العشري في هذه الحالة.

```

{
LCD_GoToLineOne();
LCD_DisplayString("Near Hot Temp.");
itoa(tempC,display,10);
LCD_GoToLineTwo();
LCD_DisplayString(display);
LCD_DisplayString(".");
itoa(tempC%10,display,10);
LCD_DisplayString(display);
PORTD=0x03;

}

else if((tempC>25)&(tempC<35))
{
LCD_GoToLineOne();
LCD_DisplayString("Very Nice Temp.");
itoa(tempC,display,10);
LCD_GoToLineTwo();
LCD_DisplayString(display);
LCD_DisplayString(".");
itoa(tempC%10,display,10);
LCD_DisplayString(display);
PORTD=0x04;

}

else if((tempC>20)&(tempC<25))
{
LCD_GoToLineOne();
LCD_DisplayString("Near Cold Temp.");
itoa(tempC,display,10);
LCD_GoToLineTwo();
LCD_DisplayString(display);
}

```

```

LCD_DisplayString(".");
itoa(tempC%10,display,10);
LCD_DisplayString(display);
PORTD=0x06;
}
else if(tempC<20)
{
LCD_GoToLineOne();
LCD_DisplayString("Very Cold Temp.");
itoa(tempC,display,10);
LCD_GoToLineTwo();
LCD_DisplayString(display);
LCD_DisplayString(".");
itoa(tempC%10,display,10);
LCD_DisplayString(display);
PORTD=0x02;
}
}

```

لاحظ في شكل ١٧-٧ أن درجة الحرارة على الشاشة تتوافق تقريباً مع الدرجة التي يقرأها الحساس مع اختلاف بسيط نتيجة تفريغ الكسور في عملية معايرة خرج المحول التماثلي الرقمي للتتوافق قراءته مع درجة الحرارة القادمة من الحساس.

## ٩-٧ مقترن مشروع تطبيقي للتنفيذ

مطلوب في هذا المشروع قراءة درجة حرارة الغرفة باستمرار، وعلى ضوء هذه الحرارة يتم التحكم في سرعة مروحة بحيث تعمل المروحة بصورة آلية، بمعنى أنها مثلاً عندما تكون درجة الحرارة أقل من ٢٥ درجة مئوية، فإن المروحة لا تعمل، وعندما تكون درجة الحرارة بين ٢٥ و ٣٠ درجة مئوية تعمل المروحة على السرعة الأولى (البطيئة)، وعندما تكون درجة الحرارة بين ٣٠ و ٣٥ درجة، تعمل المروحة عند السرعة الثانية (المتوسطة)، وعندما تكون درجة الحرارة أعلى من ٣٥ درجة مئوية تعمل المروحة عند السرعة الثالثة (العلية).

بذلك ننهى هذا الفصل عند هذا الحد وبالطبع سنستخدم المحول التماثلى الرقمى فى الكثير من التطبيقات القادمة فى الفصول التالية.

## **ملخص الفصل**

بدأ الفصل ببعض التعريفات المهمة للمحول التماثلى الرقمى والتى ننصح بقراءتها قبل والاستزادة منها من أى مصادر أخرى نظراً لأهميتها لفهم عمل هذا النوع من المحولات. بعد ذلك تم شرح مواصفات المحول التماثلى الرقمى الموجود في المتحكم atmega328، ثم شرح لطريقة تشغيل المحول من خلال دراسة تفصيلية لبيانات المسجلات المستخدمة في تشغيله.

# الفصل ٨



## المؤقتات ... المؤقت ٠

## Timers ... Timer 0

**العناوين المضيئة في هذا الفصل:**

- ١- مصادر نبضات تزامن المؤقتات
- ٢- سجل المؤقت صفر
- ٣- سجل مقارنة الخرج A
- ٤- سجل مقارنة الخرج B
- ٥- مقاطعة المؤقت صفر
- ٦- مسجلات التحكم في أداء المؤقت صفر
- ٧- حالات تشغيل المؤقت صفر

## ١-٨ مقدمة

**تعتبر** المؤقتات أو العدادات من الملحقات المهمة والأكثر استخداماً في جميع المتحكمات بلا استثناء تقريباً، فقليلًا ما تجد متحكم لا يحتوى على مؤقت أو عداد واحد على الأقل، مهما كان نوعه أو الشركة المصنعة له، وذلك لكثره التطبيقات التي تستخدم هذه المؤقتات. أولاً:  **علينا أن نفهم الفرق بين المؤقت والعداد**. المؤقت هو كيان (دائرة أو برنامج) يعطيك أزمنة تأخير بقيم مختلفة. فإنه مثلاً يعطيك إشارة بعد مرور زمن معين يتراوح من الصغر إلى عدد قليل من المايكلروثانية، ومن الكبر بما يصل إلى أيام. ومن الممكن استخدام هذه المؤقتات في الكثير من التطبيقات مثل قياس تردد إشارة معينة، أو قياس عرض نبضة من النبضات، أو قياس سرعة موتور، أو إنتاج نغمات موسيقية، أو تشغيل نظام الإشعال في السيارة ignition system، وهكذا يمكن أن تخصي الكثير من التطبيقات.

عند استخدام المؤقت كعداد، فإنه يقوم بحساب عدد النبضات الداخلة على أحد أطراف المتحكم، فمثلاً يمكن حساب عدد الكراتين أو العلب المارة على أحد خطوط الإنتاج بعد تحويلها إلى نبضات كهربية كما سنرى في بعض المشاريع التي سيتم تقديمها في هذا الفصل.

### كيفية الحصول على أزمنة التأخير

يمكن الحصول على أزمنة التأخير بطريقتين، إما برمجيا software، أو عن طريق المكونات hardware. في الطريقة البرمجية يتم عمل حلقات لا يتم عمل شيء بداخلها مثل:

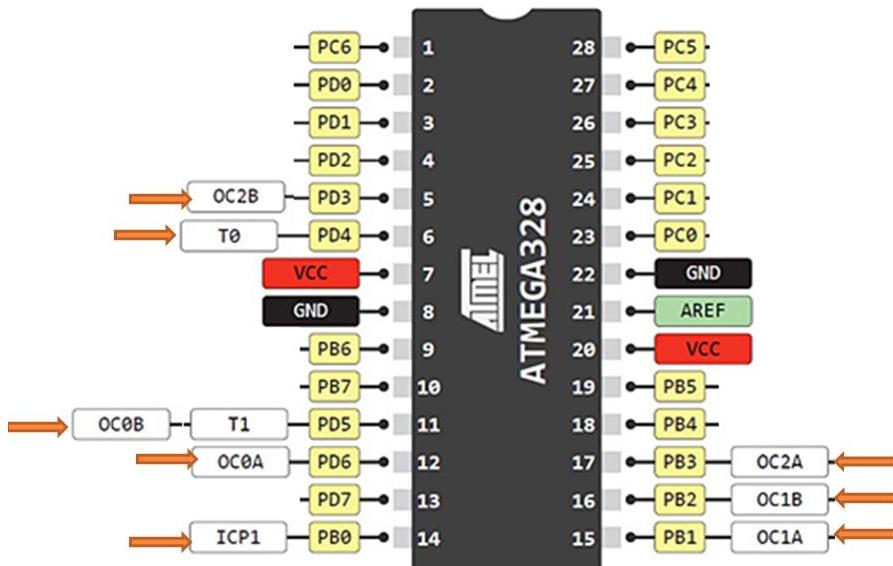
```
a=0;  
while(a<100) a++;
```

حيث هنا سيتم وضع المتغير  $a=0$  ثم البدأ في زيادة قيمته بمقدار واحد داخل الحلقة while ، حتى إذا وصلت قيمة  $a=100$  فإنه يخرج من الحلقة. زمن التأخير هنا سيساوى ١٠٠ مرة زمن تنفيذ الأمر  $a++$  بالإضافة إلى زمن تنفيذ أمر while . ويمكن التحكم في مقدار زمن التأخير عن طريق القيمة العظمى لزمن التأخير، أو عمل حلقات متداخلة بحيث يمكن عن طريق ذلك الوصول إلى أزمنة تأخير كبيرة جداً. العيب الأساسي في هذه الطريقة هي اعتمادها على نبضات ساعة المتحكم أو المعالج المستخدم، فالمتحكمات ذات الساعة القليلة ستعطى زمن تأخير كبير، والمتحكمات ذات الساعة الكبيرة ستعطى زمن تأخير قليل. وبذلك لا يمكن الاعتماد على هذه الطريقة في تنفيذ التطبيقات التي يتم تداولها بين الأجهزة المختلفة والتي يكون كل منها له نبضات الساعة الخاصة به، حيث ستكون استجابة نفس التطبيق مختلفة من جهاز آخر.

الطريقة الثانية للحصول على أزمنة التأخير هي من خلال وضع مذبذب مخصوص بتردد ثابت والمعروف يتم استخدام تردداته عن طريق دوائر أو حتى برامج أزمنة التأخير بحيث لا تكون قيمة زمن التأخير مرتبطة بتردد ساعة المعالج أو المتحكم الأساسي في الجهاز. من أمثلة ذلك المذبذبات الموجودة في كل أجهزة الحاسوب ذات التردد الثابت (١٠ ميجاهرتز تقريباً) والتي يتم استخدام تردداتها في حساب أزمنة الزمن الحقيقي (اليوم والساعة والتاريخ) وأزمنة التأخير. وهذا هو السبب أن الكثير من التطبيقات التي تستخدم أزمنة التأخير (الألعاب مثلاً) يتم استخدامها على كل أجهزة الحاسوب بدون أي مشاكل أو تغير في أدائها.

المؤقتات في كل المتحكمات تقريباً تعتمد في عملها على عدد نبضات تزامن معينة، يختلف مصدرها من متحكم لآخر كما سنرى، وعندما يصل هذا العدد لقيمة معينة تتوقف عملية العد ويتم إعطاء إشارة بذلك حيث سيكون زمن التأخير في هذه الحالة هو زمن نبضة تزامن واحدة في عدد هذه النبضات التي تم الحصول عليها.

**المتحكم atmega328 يحتوى على ثلاثة مؤقتات**، اثنان منها وهما المؤقت صفر Timer0 و المؤقت ٢ Timer2 كل منهما ٨ بت، بمعنى أن أقصى عدد نبضات يمكن أن يعدها هو ٢٥٦ أو ٨ أس ٢. المؤقت الثالث هو المؤقت ١ Timer1 وهو مؤقت ١٦ بت بمعنى أنه يمكن أن يعده حتى ٦٥٥٣٦ أس ١٦، وسنرى تفاصيل عمل كل واحد من هذه المؤقتات في هذا الفصل والفصلين التاليين مع التمررين عليها بالطبع.



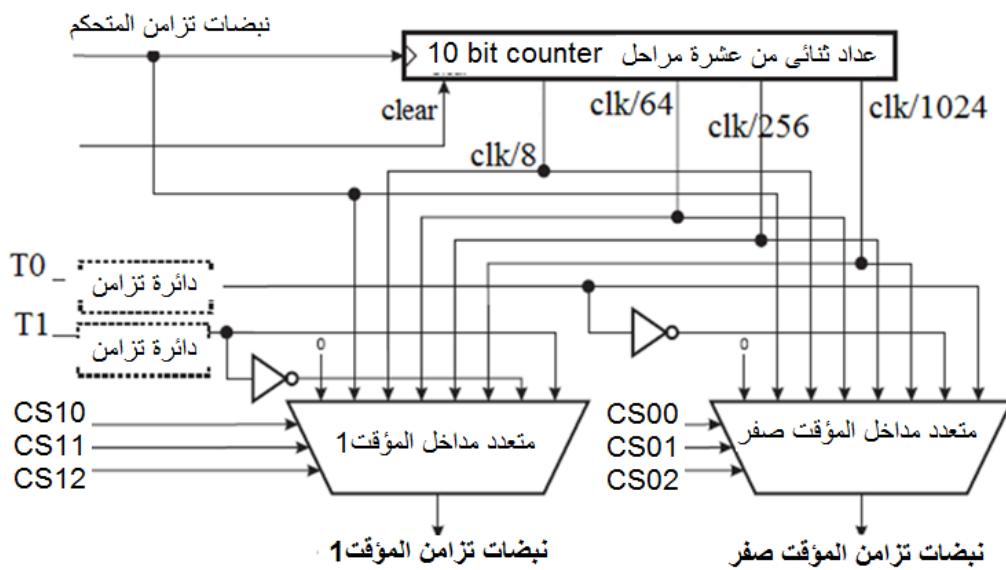
شكل ١-٨ أطراف المتحكم atmega328 المستخدمة مع المؤقتات

شكل ١-٨ يبين أطراف المتحكم atmega328 المستخدمة مع المؤقتات الثلاثة في هذا المتحكم والتي سندرس دور ووظيفة كل منها بالتفصيل.

### ٣-٨ مصادر نبضات تزامن المؤقتات

يمكن للمؤقتات أن تحصل على نبضات التزامن بترددات مختلفة إما خارجيا من على أطراف شريحة المتحكم، أو داخليا من مراحل قسمة عديدة لنبضات التزامن الأساسية للمتحكم. يتم استخدام متعدد مداخل multiplexer لاختيار أحد هذه المصادر من خلال باتات في أحد المسجلات كما سنرى. شكل ٢-٨ يبين هذه المصادر المختلفة المستخدمة مع المؤقتين Timer0 ذو الثمانى باتات، والمؤقت Timer1 ذو الستة عشرة بات، حيث يتشارك كل من هذين المؤقتين في نفس هذه الدائرة. المكونات الأساسية في هذه الدائرة هي:

- ١ - اثنان من متعددات المداخل multiplexer أحدهما يستخدم خرجه كمصدر لنبضات التزامن للمؤقت صفر، والثانى يستخدم خرجه كمصدر لنبضات التزامن للمؤقت ١. كل من متعدد المداخل له ٨ مدخل وثلاث خطوط لاختيار واحد من هذه المداخل وتوصيله على الخرج. المدخل الثمانى لكل متعدد هو ٤ مدخلقادمة من ناتج القسمة باستخدام العداد الثنائى، واثنان من الدخل الخارجى T0 ومعكوسه، أو T1 ومعكوسه، ودخل يحمل نبضات تزامن المتحكم الأساسية بدون قسمة، ودخل يمثل صفر، وهذا يعني أن المؤقت لن يعمل في هذه الحالة.
- ٢ - خطوط التحكم CS00 و CS01 و CS02 بالنسبة للمؤقت صفر، و CS10 و CS11 و CS12 بالنسبة للمؤقت ١، هى باتات أحد المسجلات التي سنتكلم عنها بالتفصيل بعد قليل.
- ٣ - عداد ثنائى binary counter مكون من عشرة مراحل يتم استخدام خرج بعض مراحله كمعامل قسمة لتزداد نبضات الدخل الذى هو نبضات التزامن الأساسية للمتحكم. فتى هناك مخارج تمثل معاملات القسمة على ٨ و ٦٤ و ٢٥٦ و ١٠٢٤.
- ٤ - المدخلان T0 و T1 هما المصادر الخارجية لنبضات التزامن التي يمكن توصيلها على الطرفين ٦ و ١١ على شريحة المتحكم على الترتيب وكما هو موضح في شكل ١-٨. هذه النبضات الخارجية يتم إدخالها على دائرة تزامن synchronization داخل المتحكم تعمل على أن تكون حواجز هذه النبضات متزامنة مع حواجز نبضات تزامن المتحكم. يتم استخدام هذه النبضات ومعكوسها كدخلين لكل من متعدد المداخل الأول والثانى كما في الشكل ٢-٨.



شكل ٢-٨ المصادر المختلفة لنبضات تزامن المؤقتين صفر وواحد

سنبدأ الآن في شرح المسجلات المختلفة التي يتم من خلالها التحكم في عمل هذه المؤقتات، وسنبدأ بالمؤقت صفر الذي سيكون موضوع الحديث في هذا الفصل بالكامل.

تعتمد فكرة عمل كل المؤقتات في المتحكمات AVR على وجود عدد ثانئي يبدأ العد من الصفر مع بدأ تشغيل أي واحد من المؤقتات، كما توجد هناك مسجلات للمقارنة compare register، حيث تحتوي هذه المسجلات عدداً يتناسب مع زمن التأخير المطلوب حيث يتم حساب هذا العدد بمعلومية عرض نبضة التزامن المستخدمة في المؤقت. يتم دائماً مقارنة محتويات هذه المسجلات مع خرج العداد بعد كل نبضة تزامن، وعند تساوى محتويات أي مسجل مقارنة مع محتويات العداد يتم عمل فعل معين مثل تشغيل أحد أطراف الخرج ومنها يمكن الحصول على موجة مربعة معدلة العرض pulse width modulation، PWM عند دراسة المسجلات المختلفة في كل مؤقت وسنبدأ بالمؤقت صفر ذو الثمانى بتات.

## ٤-٤ مسجل المؤقت/العداد ٠

مسجل المؤقت/العداد صفر Timer/counter0، TCNT0 يتكون من ثمانية بتات ويحتوى قيمة العداد بعد كل نبضة تزامن، ولذلك فقيمتها تتراوح بين الصفر و ٢٥٥. هذا المسجل يمكن قراءته أو الكتابة فيه في أي وقت. بعد كل

نبضة تزامن تزيد قيمة هذا العداد بمقدار واحد وتم مقارنته بمسجلات المقارنة compare register التي سيتم شرحها في الجزء التالي، وعلى ضوء هذه المقارنة يمكن اتخاذ كثيرة من الأفعال كما سنرى فيما بعد أيضاً. شكل ٣-٨ يبين رسم تخطيطياً لهذا المسجل حيث نلاحظ أنه يتكون من ٨ بิตات يمكن قراءتها وكتابتها في أي وقت.

	7	6	5	4	3	2	1	0
	TCNT0[7:0]							
:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
:	0	0	0	0	0	0	0	0

شكل ٣-٨ رسم تخطيطي لمسجل المؤقت/العداد صفر TCNT0

## ٥-٨ مسجل مقارنة الخرج A

يتكون مسجل مقارنة الخرج A للمؤقت صفر OCR0A output compare register A من ثمان بิตات أيضاً، ويتم فيه تسجيل القيمة التي سيتم مقارنتها مع محتويات مسجل المؤقت/العداد TCNT0 السابق بعد كل نبضة تزامن، وعلى ضوء هذه المقارنة، وعند تساوى القيمتين، يتم إخراج إشارة معينة على طرف خرج المقارنة A رقم ١٢ للمتحكم وهو الطرف OC0A كما في شكل ١-٨، وسيأتي تفصيل لهذا الخرج فيما بعد. شكل ٤-٨ يبين رسم تخطيطياً لهذا المسجل.

	7	6	5	4	3	2	1	0
	OCR0A[7:0]							
:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
:	0	0	0	0	0	0	0	0

شكل ٤-٨ مسجل مقارنة الخرج A للمؤقت صفر OCR0A

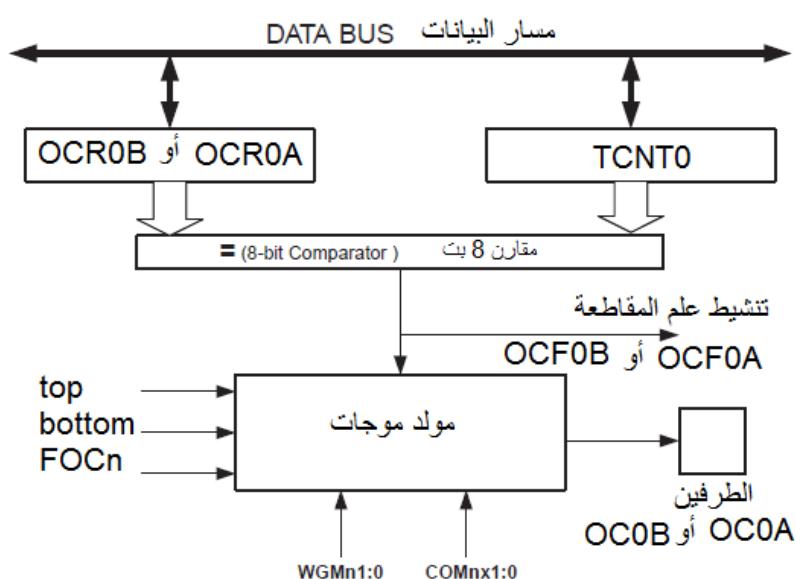
## ٦-٨ مسجل مقارنة الخرج B

يتكون مسجل مقارنة الخرج B للمؤقت صفر OCR0B output compare register B من ثمان بิตات أيضاً، ويتم فيه تسجيل القيمة التي سيتم مقارنتها مع محتويات مسجل المؤقت/العداد TCNT0 السابق بعد كل نبضة تزامن، وعلى ضوء هذه المقارنة، وعند تساوى القيمتين، يتم إخراج إشارة معينة على طرف خرج المقارنة B رقم ١١ للمتحكم وهو الطرف OC0B كما في شكل ١-٨، وسيأتي تفصيل لهذا الخرج فيما بعد. شكل ٥-٨ يبين رسم تخطيطياً لهذا المسجل.

7	6	5	4	3	2	1	0
OCR0B[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

شكل ٥-٨ مسجل مقارنة الخرج B للمؤقت صفر OCR0B

شكل ٦-٨ يبين رسمًا تخطيطيًا لعملية مقارنة الخرج في المؤقت صفر. نلاحظ من هذا الشكل أن هناك مقارن ٨ بت يقوم بمقارنة بثبات مسجل المؤقت/العداد TCNT0 مع بثبات مسجل المقارنة OCR0A أو OCR0B (حيث  $x$  يقصد بها A أو B) بحيث أنه عند تساوي القيمتين فإنه إما أن يتم تنشيط علم مقاطعة خاص بالمقارنة A أو المقارنة B وهو OCF0B وهو تميدها للانتقال إلى برنامج خدمة مقاطعة إذا كانت هذه المقاطعة فعالة وكان علم المقاطعة العام في مسجل الأعلام فعالًا أيضًا، أو أنه يتم إخراج موجة معدلة العرض PWM على أي من الطرفين OC0B أو OC0A من أطراف شريحة المتحكم كما أشرنا مسبقًا.



شكل ٦-٨ رسم تخطيطي لعملية المقارنة في المؤقت صفر

## ٧-٨ مقاطعة المؤقت 0

المؤقت صفر له ثلاثة مصادر للمقاطعة كما هو مبين في الجدول ١-٨ الذي يوضح مصدر كل مقاطعة، والعنوان الذي يتم القفز إليه عند حدوث المقاطعة، ورقم متوجه المقاطعة. المقاطعة الأولى تقع عند حدوث تساوي بين محتويات

مسجل المؤقت TCNT0 ومسجل مقارنة الخرج OCR0A. عند وقوع هذا التساوى بشرط أن يكون قناع المقاطعة الخاص بهذه المقاطعة نشطا، وأن يكون علم المقاطعة العام في مسجل الأعلام نشطا، فإن المتحكم سيقفز إلى العنوان TCNT0 كما في الجدول ١-٨ . المقاطعة الثانية تقع عند حدوث تساوى بين محتويات مسجل المؤقت 0x00E ومسجل مقارنة الخرج OCR0B. عند وقوع هذا التساوى بشرط أن يكون قناع المقاطعة الخاص بهذه المقاطعة نشطا، وأن يكون علم المقاطعة العام في مسجل الأعلام نشطا، فإن المتحكم سيقفز إلى العنوان 0x00F كما في الجدول ١-٨ . المقاطعة الثالثة تقع عند حدوث فيضان في مسجل المرقت TCNT0 ، بمعنى أن تصل محتوياته إلى ٢٥٥ ، حيث عندها يقفز المتحكم إلى العنوان 0x010 بشرط أن يكون كل من قناع هذه المقاطعة وقناع المقاطعة العام في مسجل الأعلام نشطين.

جدول ١-٨ إسم ومصدر وعنوان ورقم متوجه المقاطعات في المؤقت صفر

رقم المتوجه	عنوان المتوجه	المصدر	إسم المقاطعة
8	0x007	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x008	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x009	TIMER2 OVF	Timer/Counter2 Overflow
11	0x00A	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x00B	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x00C	TIMER1 COMPB	Timer/Counter1 Compare Match B
14	0x00D	TIMER1 OVF	Timer/Counter1 Overflow
15	0x00E	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x00F	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x010	TIMER0 OVF	Timer/Counter0 Overflow

### مسجل أقنية مقاطعات المؤقت صفر Timer Interrupt Mask Register, TIMSK0

يبين شكل ٧-٨ بذات هذا المسجل، حيث نلاحظ استخدام أول ثلاث بذات فقط من هذا المسجل وهي كالتالى:



شكل ٧-٨ مسجل أقنية مقاطعات المؤقت صفر

**البت 0: قناع مقاطعة الفيضان Timer Over flow Interrupt Enable, TOIE**, حيث يوضع واحد في هذه البت يتم تنشيط مقاطعة الفيضان ويقفز المتحكم إلى العنوان 0x010 إذا كان علم المقاطعة العام I يساوى واحد أيضا.

**البت 1: قناع مقاطعة مقارنة الخرج Output Compare Interrupt Enable A, OCIEA**, حيث يوضع واحد في هذه البت يتم تنشيط مقاطعة تساوى العداد TCNT0 ومسجل مقارنة الخرج A ويقفز المتحكم إلى العنوان 0x00E إذا كان علم المقاطعة العام I يساوى واحد.

**البت 2: قناع مقاطعة مقارنة الخرج Output Compare Interrupt Enable B, OCIEB**, حيث يوضع واحد في هذه البت يتم تنشيط مقاطعة تساوى العداد TCNT0 ومسجل مقارنة الخرج B ويقفز المتحكم إلى العنوان 0x00F إذا كان علم المقاطعة العام I يساوى واحد.  
باقي بิตات هذا المسجل غير مستخدمة.

في كل المقاطعات الثلاث السابقة كيف يعرف المتحكم بحدوث تساوى بين مسجلات المقارنة A أو B أو حدوث فيضان في محتويات العداد. يتم ذلك عن طريق علم مقاطعة خاص بكل منها في المسجل التالي.

### مسجل أعلام مقاطعة المؤقت 0 Timer/Counter0 Interrupt Flag Register, TIFR0

يبين شكل ٨-٨ بิตات هذا المسجل، حيث نلاحظ استخدام أول ثلاثة بิตات فقط من هذا المسجل وهي كالتالي:

7	6	5	4	3	2	1	0
					OCFB	OCFA	TOV

القيم التلقائية عند بداية التشغيل

شكل ٨-٨ مسجل أعلام مقاطعات المؤقت صفر

**البت 0: علم حدوث الفيضان Timer Over flow Flag, TOV**, هذه البت (العلم) تصبح واحد عند حدوث فيضان في محتويات عداد المؤقت TCNT0، ويتم تصفيرها عن طريق المتحكم بمجرد القفز إلى برنامج خدمة المقاطعة ISR الخاص بها.

**البت 1: علم مقارنة الخرج Output Compare A Interrupt Flag, OCF0A**، هذه البت (العلم) تصبح واحد عند تساوى العداد TCNT0 ومسجل مقارنة الخرج A، ويتم تصفيرها عن طريق المتحكم بمجرد القفز إلى برنامج خدمة المقاطعة ISR الخاص بها.

**البت 2: علم مقارنة الخرج Output Compare B Interrupt Flag, OCF0B**، هذه البت (العلم) تصبح واحد عند تساوى العداد TCNT0 ومسجل مقارنة الخرج B، ويتم تصفيرها عن طريق المتحكم بمجرد القفز إلى برنامج خدمة المقاطعة ISR الخاص بها.  
باقي بثات هذا المسجل غير مستخدمة.

إذن باختصار يمكننا أن نلخص آلية المقاطعات الثلاث السابقة بأنه بمجرد حدوث تساوى بين مسجل عداد المؤقت 0 TCNT0 وأحد مسجلات المقارنة A أو B، أو حدوث فيضان في محتويات المسجل TCNT0، فإن العلم المقابل لكل منهم TOV أو OCF0A أو OCF0B يصبح واحد، وبالتالي إذا كان قناع المقاطعة المقابل لكل واحدة من هذه المقاطعات TOIE أو OCIEA أو OCIEB يساوى واحد، وإذا كان علم المقاطعة العام I في مسجل الحاله يساوى واحد أيضاً، فإن المقاطعة ستحدث، ويقفز المتحكم إلى برنامج خدمة المقاطعة ISR المقابل لأى منهم، وبمجرد انتقال المتحكم إلى برنامج خدمة المقاطعة ISR فإن العلم المقابل يتم تصفيره مرة ثانية استعداداً لحدث مقاطعة أخرى.

## ٨-٨ مسجلات التحكم في أداء المؤقت 0

يتم التحكم في أداء المؤقت 0 عن طريق مسجلين A و B لهذا الغرض، وسنعرض بثات كل من هذين المسجلين بالتفصيل في هذا الجزء. شكل ٩-٨ وشكل ١٠-٨ ي بيان رسمياً تخطيطياً لهذين المسجلين.

### مسجل التحكم A في المؤقت 0 Timer/Counter0 Control Register, TCCR0A

البتابات 0 و 1 في مسجل التحكم **TCCR0A** والبت 3 في مسجل التحكم **TCCR0B**، WGM00 و WGM01 و WGM02 : هذه البتابات الثلاثة خاصة بالتحكم في تتبع عملية العد في العداد، وفي مصدر القيمة العظمى التي يمكن أن يصل إليها العداد، وفي نوع الموجة المولدة على طرف خرج المقارنة التي يمكن استخدامها، حيث يمكن استخدام واحد من ثنائية طرق أو ثنائية حالات للتشغيل أو للحصول على موجة بشكل معين مثل الموجة المربعة

المعدلة العرض PWM كما سنرى بالتفصيل فيما بعد. جدول ٢-٨ يبين هذه الحالات الشمانية للتشغيل في مقابل قيمة كل بت من هذه البتات.

7	6	5	4	3	2	1	0
COM0A1	COM0A0	COM0B1	COM0B0			WGM01	WGM00
0	0	0	0	القيم التلقائية عند بداية التشغيل			

شكل ٩-٨ المسجل A للتحكم في المؤقت 0

جدول ٢-٨ وصف براتات حالات تشغيل المؤقت 0، لاحظ أن MAX=0xFF و Bottom=0x00

الحالة Mode	WGM02	WGM01	WGM00	حالات التشغيل Modes of operation	القيمة العظمى TOP	تجدد قيمة مسجل المقارنة عند Update OCRx at	لحظة وضع علم الفيضان الواحد TOV flag set on
0	0	0	0	العادى Normal	0xFF	فوري Immediate	MAX
1	0	0	1	موجة معدلة العرض، تصحيح الطور PWM, phase correct	0xFF	TOP	Bottom
2	0	1	0	CTC	OCRA	فوري Immediate	MAX
3	0	1	1	Fast PWM	0xFF	Bottom	MAX
4	1	0	0	محجوز، غير مستخدم	-----	-----	-----
5	1	0	1	موجة معدلة العرض، تصحيح الطور PWM, phase correct	OCRA	TOP	Bottom
6	1	1	0	محجوز، غير مستخدم	-----	-----	-----
7	1	1	1	Fast PWM	OCRA	Bottom	TOP

7	6	5	4	3	2	1	0
FOC0A	FOC0B			WGM02	CS02	CS01	CS00
0	0	0	0	القيم التلقائية عند بداية التشغيل			

شكل ١٠-٨ المسجل B للتحكم في المؤقت 0

**البنا 6 و 7 في المسجل TCCR0A:** هذه البنا COM0A0 و COM0A1 تتحكم في سلوك الخرج على طرف تساوى المقارنة OC0A، من حيث هل هذا الخط سيغير حالته مع كل تساوى بين مسجل المقارنة OCRA و المسجل TCNT0 أم أنه سيصبح صفر أم سيكون واحد. كل هذه الحالات بيبيها الجدول ٣-٨. لاحظ أن الطرف OC0A وهو الطرف ١٢ في شريحة المتحكم لابد أن يتم تشغيله كطرف خرج من خلال مسجل الاتجاه PDDR. الحالات الموجودة في الجدول ٣-٨ للطرف OC0A تكون حالات التشغيل التي لا يكون فيها تعديل موجى على عرض الموجة الخارجية PWM حيث في هذه الحالة ستعمل هذه البنا بطريقة أخرى سنشرحها بعد قليل مع شرح حالات التشغيل الثمانية.

جدول ٣-٨ حالات خرج المقارنة مع مسجل المقارنة A وفي غير حالة التشغيل PWM

		وصف الحالة
COM0A1	COM0A0	
0	0	الطرف OC0A يعمل كطرف بوابة عادي، ليس له علاقة بالمقارنة Normal port operation
0	1	عند حدوث التساوى يغير الخط OC0A من حالتة، فإذا كان صفر يصبح واحد، وإذا واحد يصبح صفر. Toggle on compare match
1	0	عند حدوث التساوى يصبح الطرف OC0A صفر، Clear on compare match
1	1	عند حدوث التساوى يصبح الطرف OC0A واحد، Set on compare match

**البنا 4 و 5 في المسجل TCCR0A:** هذه البنا COM0B0 و COM0B1 تتحكم في سلوك الخرج على طرف تساوى المقارنة OC0B، من حيث هل هذا الخط سيغير حالته مع كل تساوى بين مسجل المقارنة OCRB و المسجل TCNT0 أم أنه سيصبح صفر أم سيكون واحد. كل هذه الحالات بيبيها الجدول ٤-٨. لاحظ أن الطرف OC0B وهو الطرف ١١ في شريحة المتحكم لابد أن يتم تشغيله كطرف خرج من خلال مسجل الاتجاه PDDR. الحالات الموجودة في الجدول ٤-٨ للطرف OC0B تكون حالات التشغيل التي لا يكون فيها تعديل موجى على عرض الموجة الخارجية PWM حيث في هذه الحالة ستعمل هذه البنا بطريقة أخرى سنشرحها بعد قليل مع شرح حالات التشغيل الثمانية.

**البنا ٢ و ٣ في المسجل TCCR0A غير مستخدمة.**

### جدول ٤-٨ حالات خرج المقارنة مع مسجل المقارنة B وفي غير حالة التشغيل PWM

وصف الحالة	COM0B0	COM0B1
الطرف OC0B يعمل كطرف بوابة عادي، ليس له علاقة بالمقارنة Normal port operation	0	0
عند حدوث التساوى يغير الخط OC0B من حالته، فإذا كان صفر يصبح واحد، وإذا واحد يصبح صفر. Toggle on compare match	1	0
عند حدوث التساوى يصبح الطرف OC0B صفر، Clear on compare match	0	1
عند حدوث التساوى يصبح الطرف OC0B واحد، Set on compare match	1	1

### مسجل التحكم B في المؤقت 0 TCCR0B Control Register

البنايات ٠ و ١ و ٢ في المسجل **TCCR0B**: هذه البنايات CS00 و CS01 و CS02 يتم عن طريقها اختيار مصدر نبضات تزامن المؤقت ٠ تبعاً للجدول ٥-٨.

### جدول ٥-٨ اختيار مصدر نبضات التزامن للمؤقت ٠ (أنظر شكل ٢-٨)

الوصف	CS00	CS01	CS02
لا يوجد مصدر، ويتوقف المؤقت	0	0	0
نفس نبضات المتحكم، معامل قسمة يساوى ١ ، No prescaling	1	0	0
نبضات المتحكم على ٨	0	1	0
نبضات المتحكم على ٦٤	0	1	1
نبضات المتحكم على ٢٥٦	1	0	0
نبضات المتحكم على ١٠٢٤	1	0	1
مصدر النبضات هو الطرف T0 ويعمل على الحافة النازلة	0	1	1
مصدر النبضات هو الطرف T0 وي العمل على الحافة الصاعدة	1	1	1

البت ٧ في المسجل **TCCR0B**: هذه البت FOC0A، عندما تكون واحد تسبب في إحداث مقارنة فورية بين مسجل المقارنة OCRA و المسجل TCNT0 والتأثير على طرف خرج المقارنة OC0A تبعاً للأطراف COM0A0 و COM0A1 الموضحتين في جدول ٣-٨. هذه البت لا تعمل إلا مع حالات التشغيل التي لا يكون فيها تعديل لعرض الموجة PWM، أى أن يكون مفعلاً عن طريق بات شكل موجة التعديل WGM، لذلك يجب أن تكون هذه البت صفر في حالة العمل في هذه الحالات من التشغيل.

**البت ٦ في المسجل TCCR0B:** هذه البت FOC0B، عندما تكون واحد تسبب في إحداث مقارنة فورية بين مسجل المقارنة OCRB و المسجل TCNT0 والتأثير على طرف خرج المقارنة OC0B تبعا للأطراف COM0B1 و COM0B0 الموضحتين في جدول ٤-٨. هذه البت لا تعمل إلا مع حالات التشغيل التي يكون فيها تعديل لعرض الموجة PWM، أى أن يكون معطلا عن طريق بتات شكل موجة التعديل WGM، لذلك يجب أن تكون هذه البت صفر في حالة العمل في هذه الحالات من التشغيل.

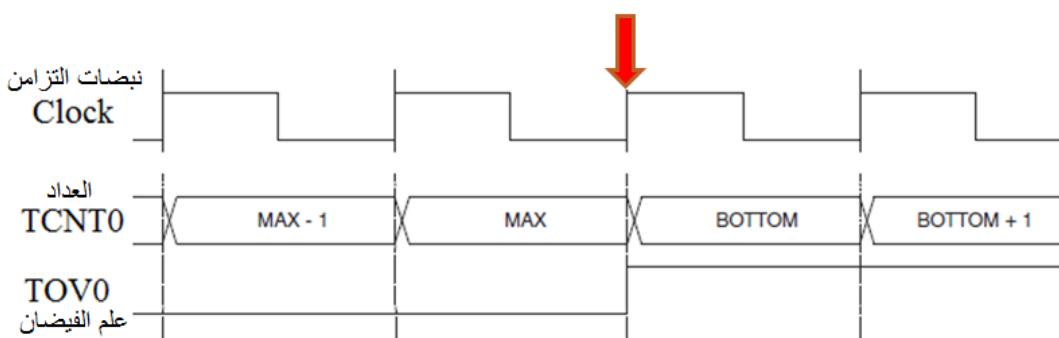
**البتات ٤ و ٥ في المسجل TCCR0B: غير مستخدمة.**

## ٩-٨ حالات تشغيل المؤقت ٠

تحدد حالات التشغيل، أو أداء المؤقت على أطراف المقارنة OC0A أو OC0B على ضوء بتات اختيار الشكل الموجى على هذه الأطراف WGM في المسجلين TCCR0A و TCCR0B ، والبتات COM0A0 و COM0A1 في المسجل TCCR0A . هناك ثمانية من هذه الحالات التي سنقدمها في هذا الجزء والتي تم عرضها في الجدول ٢-٨ .

### ١- حالة التشغيل العادية Normal mode

هذه الحالة من حالات التشغيل هي أبسط حالة في الحالات الثمانية. يتم الدخول في هذه الحالة بوضع بتات شكل الموجة WGM00 و WGM01 و WGM02 كلها بأصفار كما في الجدول ٢-٨ . في هذه الحالة يقوم العداد TCNT0 بالعد دائما في الاتجاه التصاعدى بزيادة مقدارها واحد، ولا يتم تصفير العداد نهائيا إلا عند وصوله للقيمة العظمى TOP=0xFF ، حيث عندها يبدأ العداد في العد مرة ثانية من القيمة الصغرى Bottom=0x00 . بمجرد أن تصبح قيمة العداد TCNT0 تساوى صفر ومع نفس نبضة التزامن فإن علم الفيضان TOV0 يصبح واحد (أى أن هذه البت تسلك مسلك البت التاسعة للعداد) ، تمهيدا للقفز إلى برنامج خدمة المقاطعة ISR إذا كان قناع هذه المقاطعة وعلم المقاطعة العام نشطين كما أشرنا مسبقا. هذا العلم يتم تصفيره بمجرد القفز إلى برنامج خدمة المقاطعة ISR تمهيدا لدورة العد التالية. شكل ١١-٨ يبين التزامن بين نبضات التزامن وقيمة العداد TCNT0 وعلم الفيضان، حيث نلاحظ أنه بمجرد وصول العداد للقيمة العظمى (السهم الأحمر) يرتفع علم الفيضان من الصفر إلى الواحد كما أشرنا.



شكل ١١-٨ التزامن بين نبضات التزامن والعداد TCNT0 وعلم الفيضان في حالة التشغيل

### مثال على التشغيل العادي Normal mode للمؤقت 0

برنامج يضع المؤقت 0 في الحالة العادية بحيث عندما يحدث فيضان  $TOV0=1$  فإن المتحكم يقفر إلى برنامج خدمة مقاطعة الفيضان ويجمع واحد على محتويات البوابة D التي يتم عرض محتوياتها على دايوهات ضوئية تعكس عدد مقاطعات الفيضان التي تحدث ثنائيا. شكل ١٢-٨ يبين الدائرة المستخدمة في برنامج بروتس، والبرنامج الذي سينفذ ذلك هو:

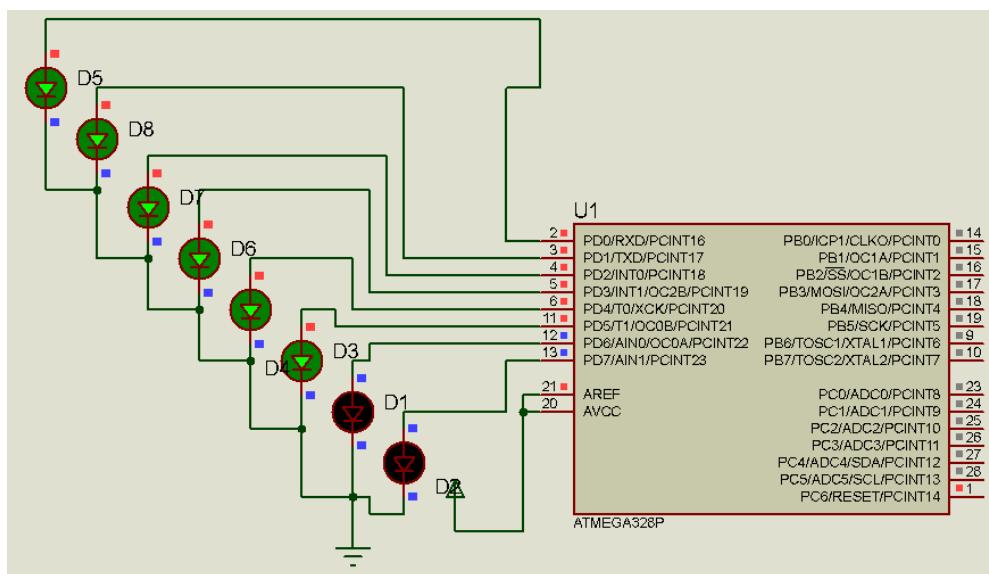
```
/*
 * Timer1.c
 * Timer 0 is in the normal mode
 * Created: 8/10/2017 9:44:08 AM
 * Author : Mohamed Eladawy
 */
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 1000000

int main(void)
{
    DDRD=0xFF; // set PD as output port
    TCCR0A =0x00;// set timer mode to normal
    TIMSK0 |= (1 << TOIE0); // enable overflow interrupt mask
```

```

sei();
TCCR0B |= (1 << CS02) | (1<<CS00); // set prescalar to 1024
while (1)
{
}
ISR (TIMER0_OVF_vect) // ISR for timer0 Over flow
{
    PORTD++;
}

```



شكل ١٢-٨ عدد مرات الفيضان في المؤقت 0 من خلال طريقة التشغيل العادي

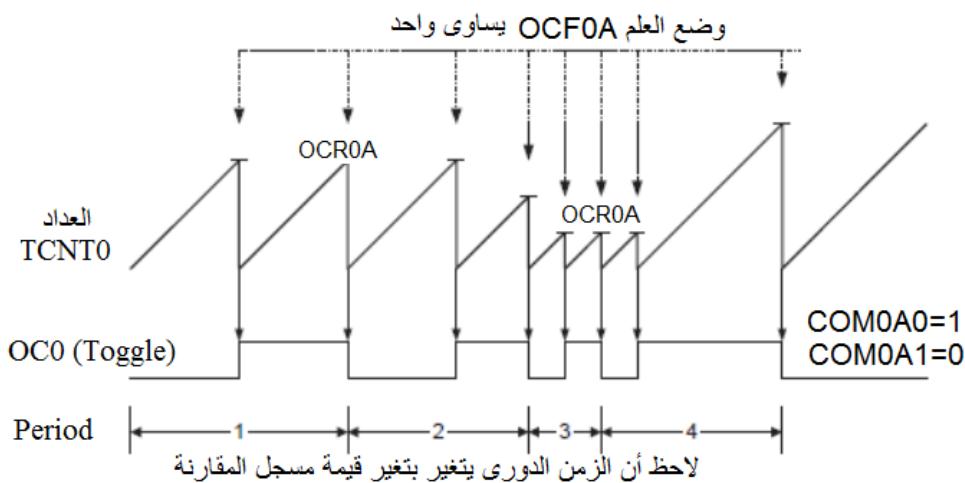
## ٤- تصفيير المؤقت 0 عند تساوى المقارنة CTC

في هذه الحالة، تصفيير المؤقت عند تساوى المقارنة، CTC، تكون الشفرة على بثات توليد الموجة WGM00 و WGM02 و WGM01 تساوى الرقم ٠١٠، كما في الجدول ٢-٨ . يتم استخدام مسجل المقارنة OCR0A لتوضع فيه القيمة التي سيتم مقارنتها مع محتويات العداد TCNT0 . في هذه الطريقة عندما يحدث التساوى بين الاثنين يتم تصفيير العداد ليبدأ دورة عد جديدة. معنى ذلك أن قيمة مسجل المقارنة OCR0A تكون

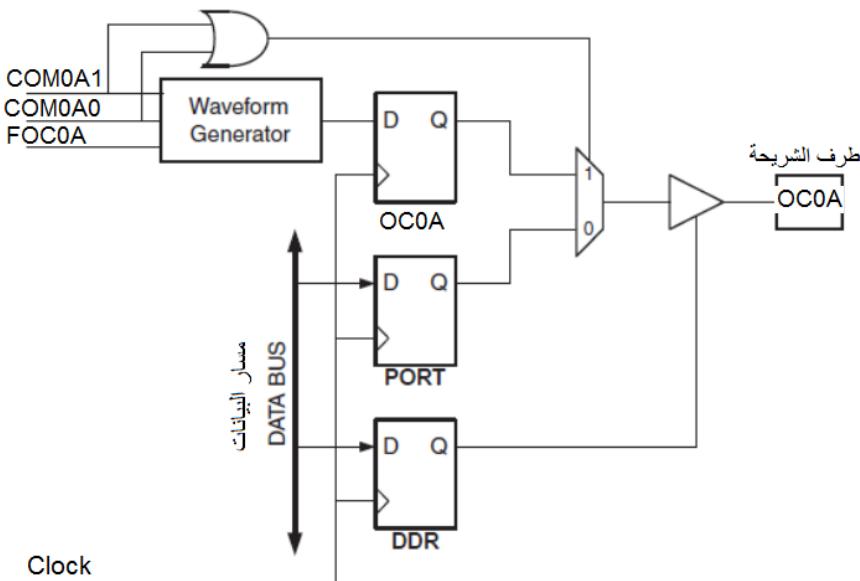
هي القيمة العظمى للعداد، وعند حدوث هذا التساوى فإن المقاطعة OCF0A يصبح واحد، وبالتالي فإن المتحكم يمكن أن يقفر إلى برنامج خدمة المقاطعة ISR الخاص بهذا الحدث إذا كان كل من القناع الخاص بهذه المقاطعة OCIE0A وعلم المقاطعة العام I نشطين، وبالتالي فإن علم المقاطعة OCF0A سيعود إلى الصفرة مرة ثانية بمجرد الدخول في برنامج خدمة المقاطعة. شكل ١٣-٨ يبين مخطط التزامن في هذه الحالة حيث نلاحظ أنه بمجرد وصول العداد TCNT0 إلى القيمة الموجودة في المسجل OCR0A فإن العداد يتم تصفيه. يمكن استخدام هذه الحالة في الحصول على شكل موجى متغير على الطرف OC0A عن طريق تعديل القيمة الموجودة في مسجل المقارنة في برنامج خدمة المقاطعة مع وضع البิต COM0A1 بحيث يغير هذا الطرف من حالته (toggle) عند كل تساوى، كما في جدول ٣-٨. لاحظ أن الإشارة على الطرف OC0A لن تكون مرئية إلا إذا كان هذا الطرف موضوع كطرف خرج عن طريق مسجل الاتجاه الخاص بالبوابة التابع لها هذا الطرف. يمكن تحديد تردد الموجة الناشئة على الطرف OC0A تبعاً للمعادلة التالية:

$$f_{OC0A} = \frac{f_{clock}}{2N(1+OCR0A)}$$

حيث N هي معامل القسمة (١ أو ٨ أو ٦٤ أو ٢٥٦) المستخدم للحصول على نبضات تزامن المؤقت،  $f_{clock}$  هي نبضات تزامن المتحكم. لاحظ أن أكبر تردد يمكن الحصول عليه في هذه الحالة سيكون عندما  $OCR0A=0$ .



شكل ١٣-٨ التزامن على الطرف OC0A مع قيمة مسجل المقارنة OCR0A (نسخة atmega328 من دليل المتحكم)



شكل ٤-٨١ علاقة طرف المقارنة OC0A مع طرف البوابة المقابل له (نسخة من دليل المتحكم (atmega328

شكل ٤-٨١ يبين العلاقة بين خرج المقارنة OC0A (ومثله تماماً الخرج OC0B) والبوابة التي هو أحد أطرافها. نلاحظ من هذا الشكل أن الطرف OC0A (في أقصى يسار الشكل) لكي يعمل هذه الوظيفة فإن خرج قلاب الاتجاه DDR يجب أن يكون واحد حتى ينশط العازل buffer الموصل لهذا الطرف بالخرج القادر من القلاب في أعلى الشكل. في هذه الحالة إذا كان أي واحد من البتات COM0A0 أو COM0A1 أو COM0A1 أو كليهما يساوى واحد فإن بوابة الأور ستعطي واحد وهذا الواحد سيمرر الدخل الأعلى (وهو الإشارة القادمة من مولد الموجات wave form generator) من متعدد المداخل إلى طرف الخرج OC0A. على النقيض من ذلك، إذا كان خرج بوابة الأور يساوى صفر (كل من COM0A0 و COM0A1 يساوى صفر في نفس الوقت)، فإن الدخل الأسفل من متعدد المداخل وهو خرج البوابة سيوصل على طرف الخرج، وفي هذه الحالة فإن الطرف OC0A لن يؤدى وظيفته، ولكنه سيصبح طرف خرج عادي من أطراف البوابة D. لذلك كان التأكيد فيما سبق على أنه لكي يؤدى طرف المقارنة OC0A أو OC0B وظيفته، فإن طرف البوابة المقابل يجب أن يتم تعينه على أنه طرف خرج بوضع واحد في قلاب الاتجاه المعاكس.

## مثال على طريقة تشغيل تصفيير المؤقت عند تساوى المقارنة، CTC

في هذا المثال يتم تشغيل المؤقت ٠ في طريقة التصفير عند التساوى CTC، مع وضع قيمة معينة في مسجل المقارنة OCR0A بحيث عندما تتساوى قيمة العداد مع القيمة الموجودة في مسجل المقارنة يتم القفز إلى برنامج خدمة مقاطعة ISR. في برنامج خدمة المقاطعة سيتم زيادة محتويات مسجل المقارنة بحيث نغير من زمن الخرج على الطرف OC0A مع كل مرة يتم فيها القفز إلى برنامج خدمة المقاطعة، ونتيجة لذلك سنحصل على موجة معدلة التردد تقريراً كالموضحة في شكل ١٥-٨. البرنامج الذي سينفذ ذلك سيكون كالتالي:

```

/*
 * Timer2.c
 * CTC mode of operation
 * Created: 8/11/2017 7:52:32 PM
 * Author : Mohamed Eladawy
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 1000000

int main(void)
{
    DDRD=0xFF; // set PD as output port
    TCCR0A |= (1 << WGM01); // set timer mode to CTC
    TCCR0A |= (1<<COM0A0); // togle OC0A at each match
    OCR0A= 0x1F; // set the max value in OCR0A
    TIMSK0 |= (1 << OCIE0A); // enable OCIE0A interrupt mask
    sei();
    TCCR0B |= (1 << CS02) | (1<<CS00); // set prescalar to 1024

    while (1)
    {
        if(OCR0A==0x1F)

```

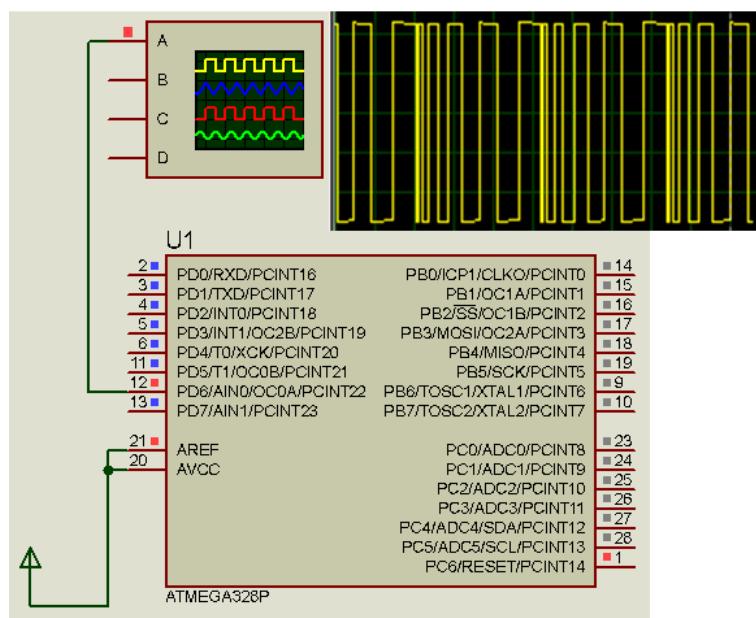
```

OCR0A=0x01;
}

}

ISR (TIMER0_COMPA_vect) // ISR for timer0 compare A
{
    OCR0A=OCR0A+3;
}

```



شكل ١٥-٨ تشغيل المؤقت ٥ في طريقة التصفيير عند تساوى المقارنة

والحصول على موجة معدلة التردد تقريباً على الطرف OC0A

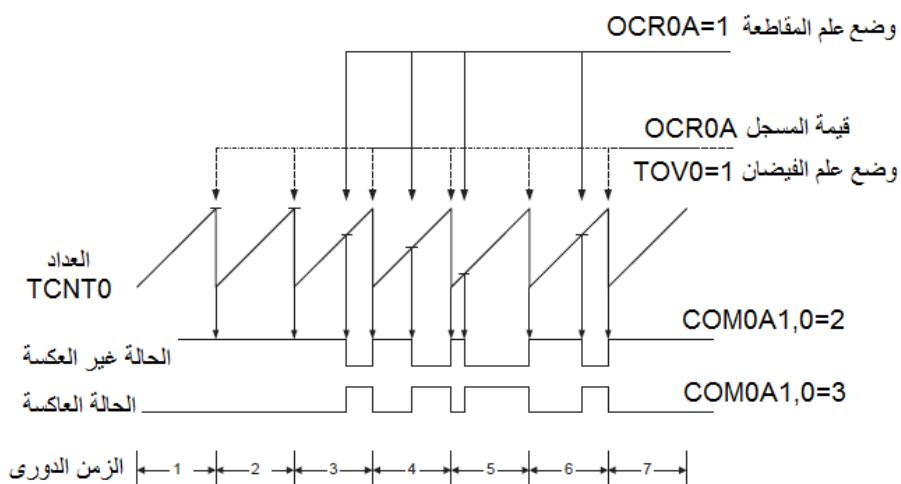
### ٣ - طريقة تعديل عرض النبضة PWM السريعة

في هذه الطريقة تكون باتات توليد الشكل الموجي تساوى ٣ (011) أو ٧ (111) كما في الجدول ٢-٨ . تتميز هذه الطريقة بوجود ميل واحد فقط وهو في حالة صعود العداد TCNT0 من صفر إلى القيمة العظمى TOP=0xFF في حالة أن باتات الشكل الموجي WGM00، WGM01، و WGM02 تساوى ٣ (011) ثم ينزل للصفر مرة ثانية ويبدأ في الصعود وهكذا. إذا كانت باتات الشكل الموجي تساوى ٧ (111)، فإن العداد يصعد من الصفر إلى

القيمة العظمى TOP=OCR0A، أى إلى القيمة المخزنة في مسجل المقارنة ثم ينزل للصفر ويبدأ في الصعود مرة ثانية، وهكذا. هناك حالتان لظهور الموجة على طرف خرج المقارنة OC0A وهما: حالة عدم العكس وفيها يتم تصفيير الطرف OC0A، عند لحظة تساوى قيمة العداد TCNT0 مع مسجل المقارنة OCR0A، ثم يتم إعادةه للواحد مرة ثانية عند يصبح العداد صفرًا. في الحالة العاكسة يتم وضع طرف خرج المقارنة بوحدة عند لحظة تساوى العداد TCNT0 ومسجل المقارنة OCR0A، ثم تصفييره عند وصول العداد للصفر. بسبب هذا الميل الوحيد وهو مع صعود العداد من الصفر للقيمة العظمى، فإن هذه الطريقة تكون أسرع (تقريباً ضعف) من الطريقة التالية وهي طريق تعديل الطور phase correct المزدوجة الميل والتي سنشرحها بعد قليل. هذه السرعة تجعل هذه الطريقة مناسبة للكثير من التطبيقات من الطريقة الثانية.

شكل ١٦-٨ يبين التزامن الحادث مع هذه الطريقة. في هذا الشكل تمثل الخطوط الأفقية القصيرة التي على إشارة العداد قيمة مسجل المقارنة التي ستتم عندها المقارنة. لاحظ في هذا الشكل أن العداد TCNT0 يعد دائمًا من الصفر إلى القيمة العظمى 0xFF، وعندما يصل العداد لهذه القيمة يبدأ من الصفر مرة أخرى ويصبح علم الفيضان TOV0 بوحدة عند هذه اللحظة. في أثناء صعود العداد TCNT0 من الصفر إلى الواحد، وفي لحظة تساويه مع قيمة مسجل المقارنة OC0A الممثلة بالخط الأفقي القصير، فإن طرف خرج المقارنة OC0A ينزل للصفر في حالة عدم العكس (البتات COM0A0=0 و COM0A1=1)، أو يصعد من صفر لواحد في حالة العكس (البتات COM0A1=0 و COM0A0=1)، ويظل الطرف OC0A على هذه الحالة إلى أن يصل العداد للحالة العظمى 0xFF حيث يرجع الطرف OC0A إلى أصله مرة ثانية، وهي الواحد في حالة عدم الانعكاس، أو الصفر في حالة العكس. إذن معنى ذلك أن الزمن الدورى للموجة الناجمة على الطرف OC0A سيكون ثابت، وأما نسبة الزمن ON للزمن OFF أثناء الزمن الدورى فسيتم التحكم فيها عن طريق القيمة الموجودة في مسجل المقارنة، ومن هنا كانت فكرة تعديل عرض الموجة PWM الموضحة في شكل ١٦-٨. في الحالة العاكسة يقل الزمن ON بزيادة القيمة المخزنة في المسجل OCR0A، وفي الحالة غير العاكسة، فإن الزمن ON يزيد بزيادة القيمة المخزنة في مسجل المقارنة OCR0A. تذكر هنا أنه لكي تتم رؤية هذه الموجات على طرف المقارنة OC0A، فإن هذا الطرف لابد أن يكون قد تم تعينه ليكون طرف خرج بوضع وحيد في مسجل اتجاه البوابة التابع لها هذا الطرف كما في شكل ١٤-٨.

لاحظ أنه في نهاية كل زمان دورى فإن علم الفيضان TOV0 يتم وضعه بوحدة، وفي هذه الحالة إذا تم تنشيط كل من قناع المقاطعة TOIE0 وعلم المقاطعة العام I، فإنه يمكن القفز إلى برنامج خدمة مقاطعة ISR، ويمكن في هذا البرنامج تعديل قيمة مسجل المقارنة OCR0A على حسب الطلب.



شكل ١٦-٨ التزامن الحادث مع طريقة تعديل عرض النبضة PWM السريع في حالة المؤقت 0 (نسخة من دليل المتحكم atmega328)

تردد الموجة المعدلة العرض PWM السريع يمكن الحصول عليها من المعادلة التالية:

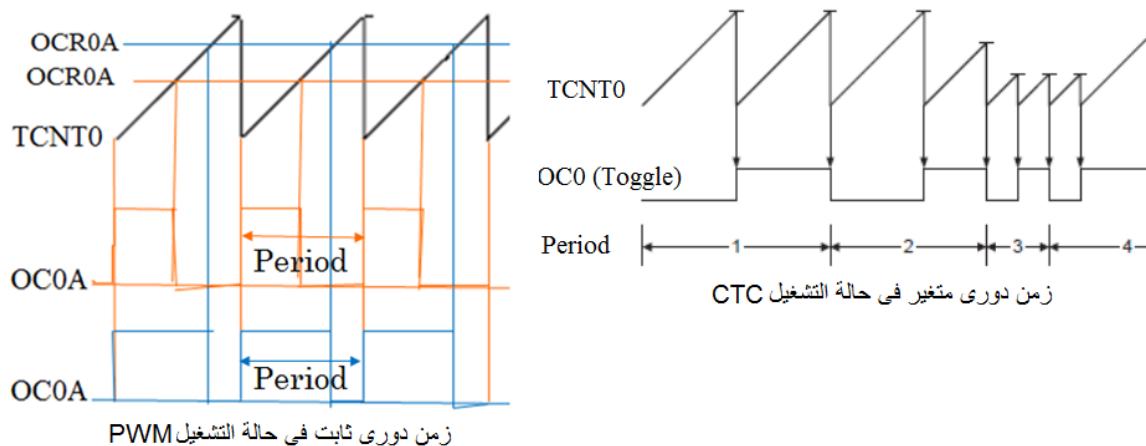
$$f_{OC0APWM} = \frac{f_{clock}}{N \cdot 256}$$

حيث  $f_{clock}$  هي نبضات تردد المتحكم، و  $N$  هي نسبة القسمة المستخدمة للحصول على نبضات التزامن التي سيعمل عندها المؤقت (١ أو ٨ أو ٦٤ أو ٢٥٦ أو ١٠٢٤). وضع القيمة العظمى 0xFF في مسجل المقارنة ستجعل الطرف OC0A يساوى واحد دائماً في الحالة غير العاكسة، أو صفر في الحالة العاكسة، وهذا هو الحال في أول دورتين في شكل ١٦-٨. وأما في حالة وضع صفر في مسجل المقارنة فإنه ستكون هناك نبضة ضيقة جداً spike ستظهر على الطرف OC0A. شكل ١٧-٨ يبين الفرق بين طريقة التشغيل CTC وطريقة التشغيل PWM حيث نلاحظ أن القيمة المخزنة في مسجل المقارنة OCR0A تتتحكم في مقدار الزمن الدورى للموجة الناتجة على الخط OC0A، بينما هذه القيمة تتتحكم في نسبة الزمن ON إلى الزمن OFF داخل كل دورة من دورات الزمن الدورى الذى يكون ثابت دائماً كما في الشكل.

### مثال على طريقة تشغيل تعديل عرض الموجة PWM السريع

البرنامج التالي يشغل المؤقت 0 في طريقة تعديل عرض الموجة PWM مع وضع الطرف OC0A ليعمل في الطريقة العاكسة، بمعنى عند تساوى محتويات العداد TCNT0 مع مسجل المقارنة OCR0A فإن الطرف OC0A يتغير

من صفر إلى واحد، وسنضع طرف خرج المقارنة OC0B ليعمل بنفس الطريقة أيضا. سنضع في المسجل OCR0A رقم كبير إلى حد ما، 0xCF، لنجعل الزمن ON على الطرف A كبيرا، وسنضع في المسجل OCR0B رقم صغير إلى حد ما، 0x0F، لنجعل الزمن ON على الطرف B صغيرا، وسنضع الإشارتين على الأوسولوسكوب لنرى الفرق بينهما كما في شكل ١٨-٨.



شكل ١٧-٨ الفرق بين الطريقة CTC والطريقة PWM في المؤقت 0

```
/*
* Timer3.c
* Fast PWM operation
* Created: 8/12/2017 2:53:59 PM
* Author : Mohamed Eladawy
*/
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 1000000
```

```
int main(void)
{
```

```

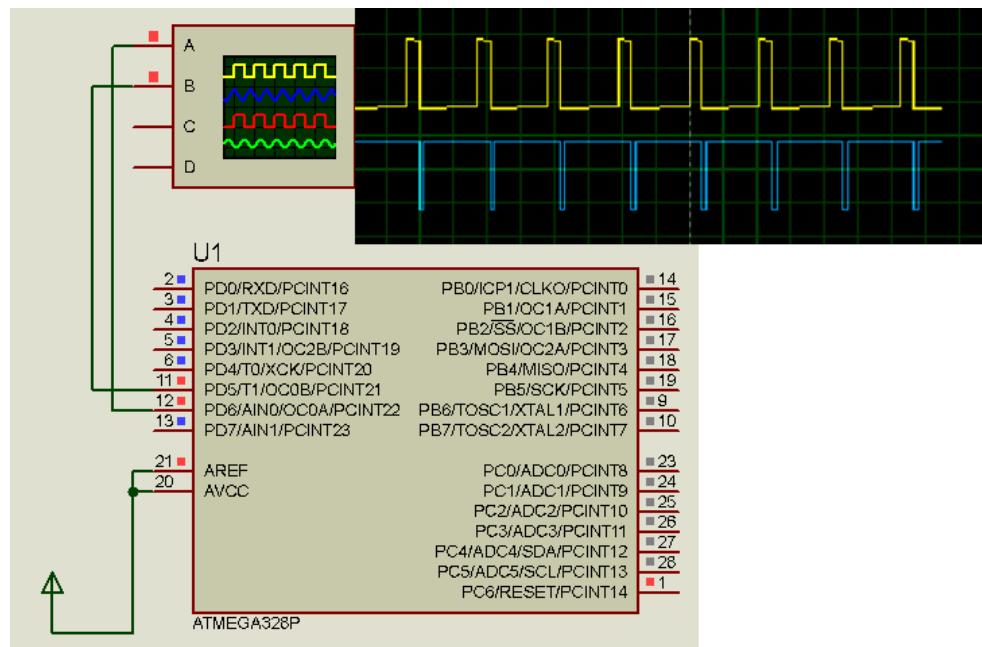
DDRD=0xFF; // set PD as output port
TCCR0A |= (1 << WGM00) | (1 << WGM01) | (1 << WGM02); // set timer
mode to PWM (7)
TCCR0A |= (1<<COM0A0) | (1<<COM0A1); // Inverting on OC0A
TCCR0A |= (1<<COM0B0) | (1<<COM0B1); // Inverting on OC0B
OCR0A= 0xCF; // set the max value in OCR0A
OCR0B= 0x0F; // set the max value in OCR0B
TCCR0B |= (1 << CS02) | (1<<CS00); // set prescalar to 1024

```

```

while (1)
{
}
}

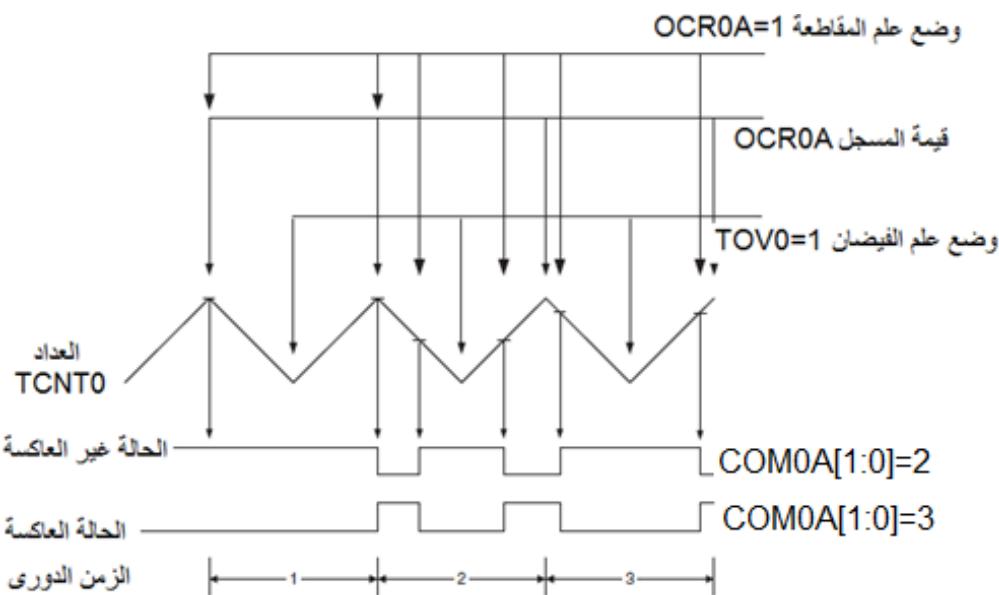
```



شكل ١٨-٨ تشغيل المؤقت ٥ بطريقة تعديل عرض النبضة PWM على كل من طرق خرج المقارنة OC0B و OC0A بقيم مقارنة مختلفة في المسجلين OCR0A و OCR0B

#### ٤- طريقة تعديل عرض النبضة PWM المعدلة الطور phase correct

في هذه الطريقة تكون برات توليد الشكل الموجي تساوى ١ (001) أو ٥ (101) كما في الجدول ٢-٨. تتميز هذه الطريقة بأنها مزدوجة الميل، أحد الميلين يكون في حالة صعود العداد TCNT0 من صفر إلى القيمة العظمى TOP=0xFF، والثانى في حالة نزول العداد بالتدرج من القيمة العظمى للصفر. في حالة أن برات الشكل الموجي WGM01، و WGM02 تساوى ١ (001) تكون القيمة العظمى هي TOP=0xFF، وإذا كانت برات الشكل الموجي تساوى ٥ (101)، فإن القيمة العظمى تكون A، TOP=OCR0A، أي أنها تكون القيمة المخزنة في المسجل OCR0A. مثل الطريقة السريعة السابقة ذات الميل الواحد، في حالة عدم العكس يتم تصفير طرف خرج المقارنة OC0A عند تساوى العداد TCNT0 مع محتويات مسجل المقارنة OCR0A في حالة الصعود من الصفر إلى القيمة العظمى، ويتم وضعه بوحدة تساوى العداد TCNT0 مع محتويات مسجل المقارنة OCR0A في حالة النزول من القيمة العظمى للصفر كما في شكل ١٩-٨. في حالة العكس يحدث عكس ذلك.



شكل ١٩-٨ التزامن الحادث مع طريقة تعديل عرض النبضة PWM المعدلة الطور

بالطبع فإن طريقة الميل المزدوج سيكون أكبر تردد لها أقل من أكبر تردد في الطريقة السريعة ذات الميل الواحد. باختصار وكما في شكل ١٩-٨ الذي يبين التزامن في هذه الحالة، فإن العداد TCNT0 يعد صاعداً من الصفر إلى القيمة العظمى، ثم ينزل تدريجياً من القيمة العظمى للصفر مرة أخرى. عند التساوى مع مسجل المقارنة في أثناء الصعود،

يصغر طرف خرج المقارنة OC0A، وعند التساوى مع مسجل المقارنة فى أثناء النزول يضع الطرف OC0A بواحد مرة أخرى.

تذكر هنا أنه لكي تتم رؤية هذه الموجات على طرف المقارنة OC0A، فإن هذا الطرف لابد أن يكون قد تم تعينه ليكون طرف خرج بعض وحيد في مسجل اتجاه البوابة التابع لها هذا الطرف كما أوضحتنا في شكل ١٤-٨.

لاحظ أنه في نهاية كل زمن دورى فإن علم الفيضان TOV0 يتم وضعه بواحد، وفي هذه الحالة إذا تم تنشيط كل من قناع المقاطعة TOIE0 وعلم المقاطعة العام I، فإنه يمكن القفز إلى برنامج خدمة مقاطعة ISR، ويمكن في هذا البرنامج تعديل قيمة مسجل المقارنة OCR0A على حسب الطلب.

تردد الموجة المعدلة العرض PWM ذات الطور المعدل يمكن الحصول عليها من المعادلة التالية:

$$f_{OC0APWM} = \frac{f_{clock}}{N \cdot 510}$$

حيث  $f_{clock}$  هي نبضات تزامن المتحكم، و N هي نسبة القسمة المستخدمة للحصول على نبضات التزامن التي سيعمل عندها المؤقت (١ أو ٨ أو ٦٤ أو ٢٥٦ أو ١٠٢٤).

### مثال على طريقة تشغيل تعديل عرض الموجة PWM ذات الطور المعدل

سنستخدم في هذا المثال مقاومة متغيرة للتحكم في سرعة موتور (تيار ثابت dc motor). شكل ٢٠-٨ يبين هذه الدائرة على برنامج البروتس. لقد تم استخدام المقاومة من خلال مقسم جهد حيث تم إدخال الجهد الخارج من مقسم الجهد على المحول التماشى الرقمي ADC الموجود داخل المتحكم، مع محاذاة نتيجة التحويل ناحية اليسار واستخدام الشمان بتات الخاصة بالمسجل ADCH لتوضع في مسجل مقارنة الخرج OC0A للتحكم في عرض النبضة الناتجة على خط مقارنة الخرج OC0A. مع تشغيل هذا الخرج بالطريقة العاكسة، فإننا سنلاحظ أنه مع وضع مقسم الجهد للحصول على أقل جهد فإن عرض الموجة سيكون أكبر ما يمكن، وزيادة الجهد مع حركة زراع المقاومة المتغيرة إلى أعلى سيزيد الجهد الداخل للمحول التماشى الرقمي، وبالتالي القيمة الموجودة في المسجل OCR0A، وبالتالي ينقص عرض الموجة. شكل ٢٠-٨ يبين عرض الموجة الناتجة عند قيمتين مختلفتين من المقاومة المتغيرة. البرنامج المستخدم في ذلك كان كالتالى:

/\*

\* Timer4.c

\* Phase correct PWM

\* Created: 8/14/2017 8:56:54 PM

\* Author : Mohamed Eladawy

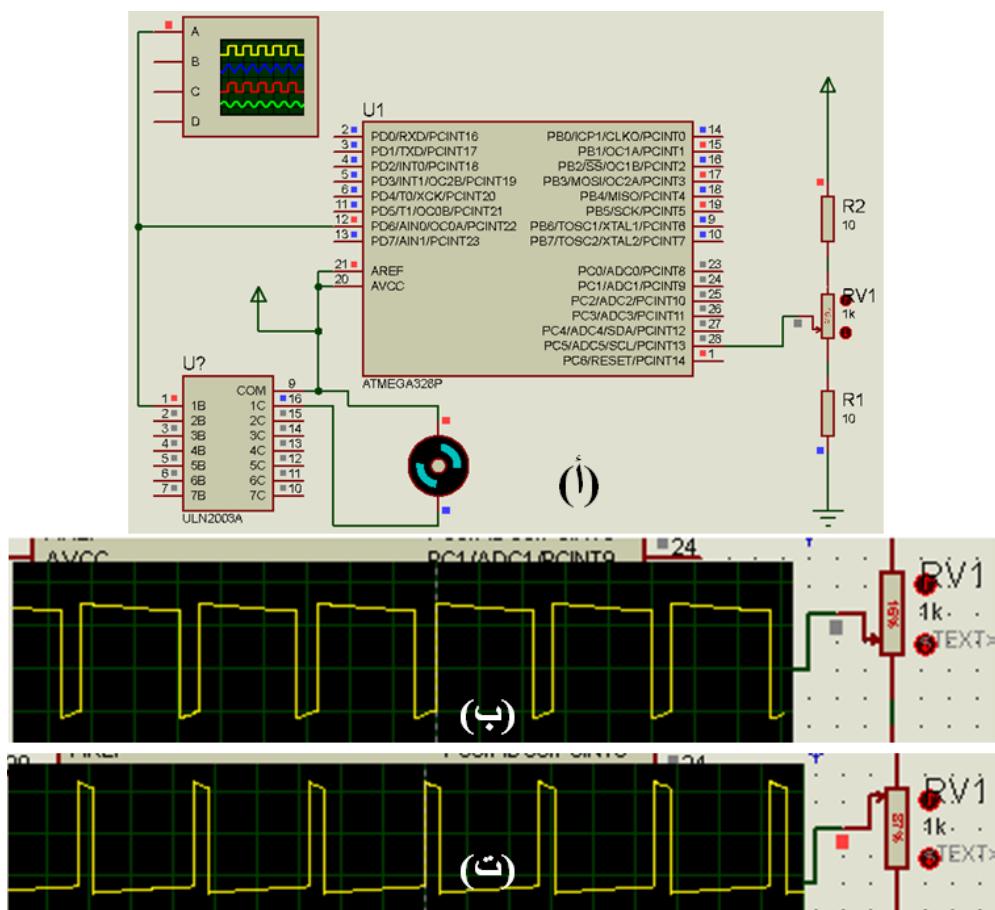
\*

```
#define F_CPU 1000000
```

```
#include <avr/io.h>
```

```
#include <avr/interrupt.h>
```

```
#include <util/delay.h>
```



شكل ٨ - تشغيل المؤقت ٠ بطريقة تعديل عرض النبضة PWM المعدل الطور لتشغيل

موتور تيار مستمر من خلال مقاومة متغيرة مع بيان عرض النسبة عند قيمتين مختلفتين

```

int main(void)
{
    DDRD=0xFF; // set PD as output port
    DDRB=0xFF; //Set PB as output port
    PORTB=0x00;
    ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS0); // enable A/D, and
    select prescalar 32
    ADMUX=0x25; // choose input channel 5, and left adjust result
    TCCR0A |= (1 << WGM00) | (1 << WGM02); // set timer mode to PWM (5)
    TCCR0A |= (1<<COM0A0) | (1<<COM0A1); // Inverting on OC0A
    TCCR0B |= (1 << CS02) | (1 << CS00); // set prescalar to 1024

    while (1)
    {
        ADCSRA |= (1<<ADSC); //start conversion
        while (ADCSRA & (1<<ADSC)); // wait for conversion to complete
        OCR0A = ADCH; // read the ADC high byte
    }
}

```

الشريحة ULIN2003A تستخدم كغازل ودفع تيار للموتور.

## ٥-تشغيل المؤقت ٥ كعداد

باختيار مصدر نبضات المؤقت من الطرف T0 للمؤقت وهو الطرف رقم ٦ في الشريحة، فإن المؤقت سيعمل كعداد يعد النبضات الداخلة على هذا الطرف. جدول ٥-٨ يبين المصادر المختلفة للنبضات، حيث باختيار البتات CS0=0 و CS1=1 و CS2=1 فإن العداد سيعمل في هذه الحالة مع الحافة النازلة للنبضات، وبوضع هذه البتات CS0=1 و CS1=1 و CS2=0، فإن العداد سيعمل مع الحافة الصاعدة للنبضات الداخلة على الطرف T0. طبعاً سيكون أقصى عدد يمكن أن يصل له العداد في هذه الحالة هو ٢٥٦ حيث أن المؤقت ٨ بت فقط.

**مثال على تشغيل المؤقت كعداد**

شكل ٢١-٨ يبين إدخال نبضات من خلال مفتاح على الطرف T0، وعرض عدد هذه النبضات على شاشة عرض ذات البللورة السائلة. البرنامج الذى سينفذ ذلك سيكون كما يلى:

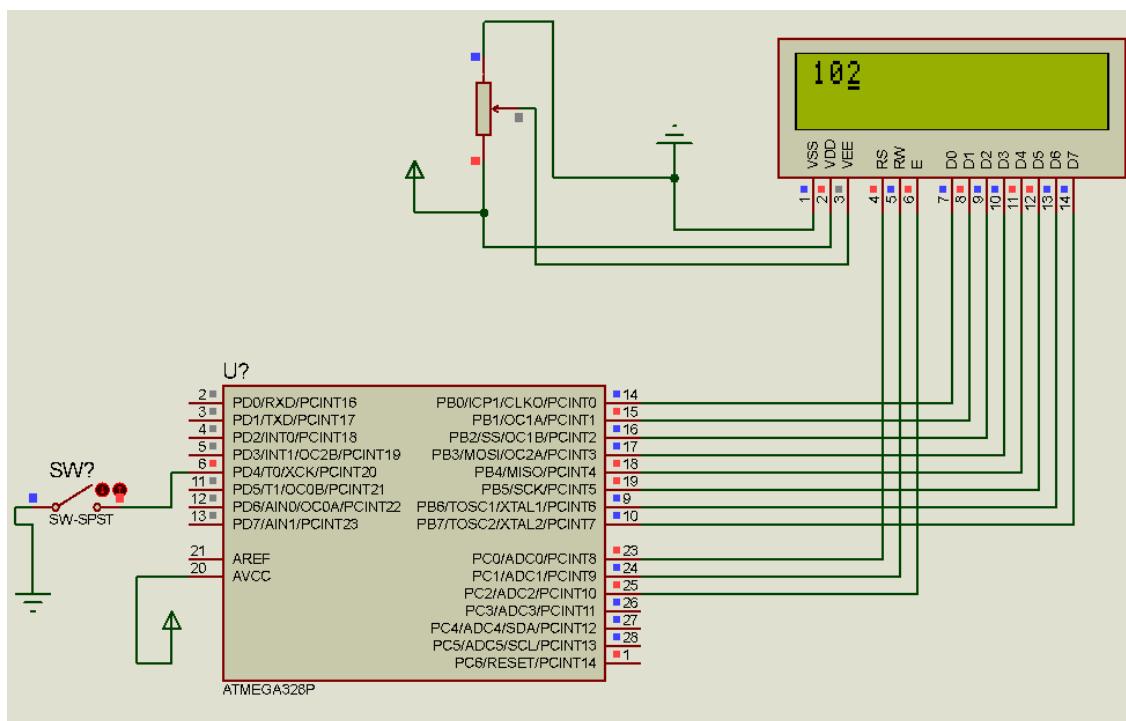
```
/*
 * Timer5.c
 * Using timer0 as a counter
 * Created: 8/17/2017 9:44:12 AM
 * Author : Mohamed Eladawy
 */
#include <avr/io.h>
#include <stdlib.h>
#include <string.h>
#define F_CPU 1000000ul
#include <util/delay.h>
#include "LCDlib.h"

int main(void)
{
    char display[5]; // الشاشة على لعرضها النتيجة تحتوى أحرف مصفوفة
    char Temp;
    DDRC =0x0F;
    LCD_Init();
    DDRD &= ~(1 << DDD4); // PD4 as input line
    PORTD |= (1 << PORTD4); //turn on Pull up resistance on this line
    DDRB=0xFF; // PB output
    TCCR0B |= (1 << CS02) | (1 << CS01) | (1 << CS00); // counter works
on rising edge
    while (1)
```

```

    {
        Temp=TCNT0;
        itoa(Temp,display,10);
        LCD_GoToLineOne();
        LCD_DisplayString(display);
    }
}
}

```



شكل ٢١-٨ تشغيل المؤقت ٠ كعداد وعرض العدد على شاشة عرض

بذلك نكون قد انتهينا من كل حالات تشغيل المؤقت ٠.

## ملخص الفصل

تناول الفصل المؤقت صفر كواحد من ثلاثة مؤقتات مهمة في المتحكم atmega328، وسيأتي شرح المؤقتات الباقيه في الفصول التالية. بدأ الفصل بتوضيح كيفية الحصول على أزمنة التأخير على وجه العموم، وكيفية الحصول على مصادر

نبضات المؤقتات، ثم ركز الفصل على المؤقت صفر من خلال دراسة مسجالاته المختلفة ووظيفة كل بت من بتات هذه المسجالات، مع أمثلة براجحية على كل طريقة من طرق تشغيل هذا المؤقت، وبالذات طريقة الحصول على نبضات معدلة العرض PWM نظراً لأهميتها في الكثير من التطبيقات وأهمها إدارة المواتير. ولا ننسى أن الفصل تناول موضوع مقاطعة المؤقت صفر للمتحكم عند حدوث ثلاثة مواقف مهمة، الأول حدوث تساوى بين مسجل المؤقت ومسجل مقارنة الخرج A، والثانى حدوث تساوى بين مسجل المؤقت ومسجل مقارنة الخرج B، والثالث حدوث فيضان فى محتويات مسجل المؤقت. عند حدوث أى واحد من هذه المواقف فإن عملية التنفيذ تنتقل إلى برنامج خدمة مقاطعة خاص بهذا الموقف ISR إذا كانت شروط هذه المقاطعة محققة.

# الفصل ٩



## المؤقتات ... المؤقت ١

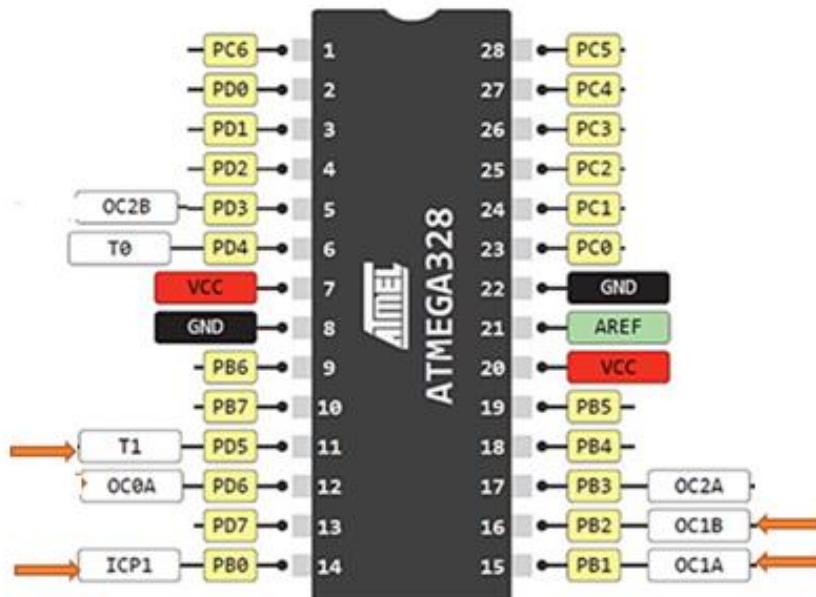
### Timers ... Timer 1

**العناوين المضيئة في هذا الفصل:**

- ١- مصادر نبضات المؤقت ١
- ٢- مسجل المؤقت ١
- ٣- مسجل مقارنة الخرج A
- ٤- مسجل مقارنة الخرج B
- ٥- مسجل مسک الدخل
- ٦- مقاطعة المؤقت ١
- ٧- مسجلات التحكم في أداء المؤقت ١
- ٨- حالات تشغيل المؤقت ١

## ١-٨ مقدمة

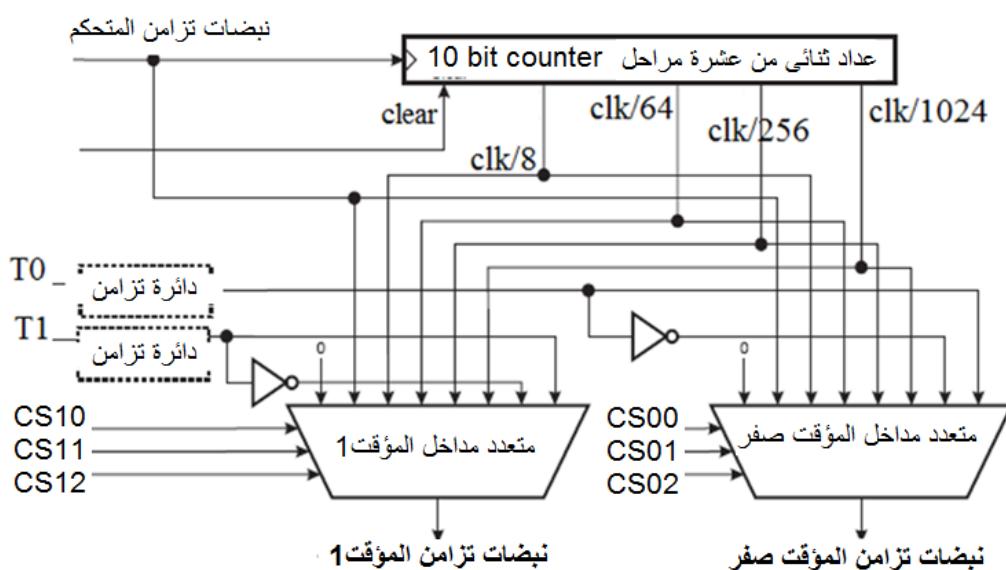
**المؤقت ١** يشبه تماماً المؤقت صفر في تشغيله مع الاختلافات البسيطة التي ستكون بسبب أن عدد هذا المؤقت مكون من ١٦ بت، وليس ثنائية مثل المؤقت صفر السابق. لذلك ستكون القيمة العظمى التي يمكن أن يصل إليها هذا العدد هي  $65536^{16}=2^{48}$ ، أو اختصاراً نقول أنها ٦٤ كيلو، وذلك على العكس من القيمة المنشورة في حالة المؤقت صفر التي كانت ٢٥٦ فقط. المشكلة التي تتوقعها هنا هي أن المتحكم atmega328 هو في الأصل متحكم ٨ بت، بمعنى أن مسار البيانات بداخله مكون من ٨ بت (خطوط) فقط، فكيف يمكن التعامل مع محتويات عداد هذا المؤقت المكونة من ١٦ بت؟ وهذا السؤال سنجيب عليه بعد قليل في هذا الفصل. بالطبع إذا كان العداد في هذا المؤقت مكون من ١٦ بت، فإن مسجلات المقارنة ستكون أيضاً ١٦ بت كما سنرى. شكل ١-٩ يبين أطراف المتحكم المستخدمة مع المؤقت ١ والتي سيأتي شرحها بالتفصيل في هذا الفصل. معظم المادة العلمية في هذا الفصل ربما تكون مكررة من الفصل السابق ولكننا فضلنا إعادتها حتى يكون الفصل مستقلاً بقدر الإمكان.



شكل ١-٩ أطراف المتحكم atmega328 المستخدمة مع المؤقت ١

## ٢-٩ مصادر نبضات التزامن للمؤقت ١

شكل ٢-٩ يبين الدائرة المستخدمة للحصول على نبضات التزامن للمؤقت ١ حيث نلاحظ أنها هي تماماً المستخدمة مع المؤقت صفر، فيما عدا أن هناك متعدد مداخل MUX منفصل للمؤقت ١ يأخذ منه نبضات التزامن الخاصة به كما أشرنا في الفصل ٨. هذه المصادر إما أن تكون من الطرف T1 الخارجي وهو الطرف ١١ لشريحة المتحكم، حيث يمكن للمستخدم أن يدخل نبضات خارجية على هذا الطرف والتي تدخل على دائرة تزامن كما في شكل ٢-٩ تعمل على توافق حواف هذه النبضات مع حواف نبضات تزامن المتحكم. باقي المصادر مأخوذة بعد قسمة نبضات تزامن المتحكم على أحد معاملات القسمة ١ أو ٨ أو ٦٤ أو ٢٥٦ أو ١٠٢٤ كما في الشكل.



شكل ٢-٩ المصادر المختلفة لنبضات تزامن المؤقت ١

ال Bates CS10 و CS11 و CS12 هي المستخدمة في اختيار أحد المدخل الثمانية لمتعدد المدخل وتوصيلها على الخرج كما في الشكل ٢-٩، وهذه Bates ستحدث عنها بعد قليل.

تعتمد فكرة عمل كل المؤقتات في المتحكمات AVR على وجود عدد ثانوي (١٦ بت في هذه الحالة) يبدأ العد من الصفر مع بدأ تشغيل المؤقت، كما توجد هناك مسجلات للمقارنة compare register، حيث تحتوى هذه المسجلات عدداً يتاسب مع زمن التأخير المطلوب حيث يتم حساب هذا العدد بعمومية عرض نبضة التزامن المستخدمة في المؤقت. يتم دائماً مقارنة محتويات هذه المسجلات مع خرج العداد بعد كل نبضة تزامن، وعند تساوى محتويات أي مسجل مقارنة مع محتويات العداد يتم عمل فعل معين مثل تشغيل أحد أطراف الخرج ومنها يمكن الحصول على موجة

مربعة معدلة العرض pulse width modulation, PWM التي يمكن استخدامها في الكثير من التطبيقات كما سنرى بالتفصيل بدراسة المسجلات المختلفة.

### ٣-٩ مسجل المؤقت ١

مسجل المؤقت ١ Timer/counter1، TCNT1 يتكون من ١٦ بت ويحتوى قيمة العداد بعد كل نبضة تزامن، ولذلك فقيمتها تتراوح بين الصفر و ٦٤٥٣٦ (٦٤ كيلو) كما ذكرنا لهذا المؤقت. هذا المسجل يمكن قراءته أو الكتابة فيه في أي وقت. بعد كل نبضة تزامن تزيد قيمة هذا العداد بمقدار واحد وتم مقارنته بمسجلات المقارنة compare register التي سيتم شرحها في الجزء التالى والتي يتكون كل منها من ١٦ بت أيضاً، وعلى ضوء هذه المقارنة يمكن اتخاذ كثيراً من الأفعال كما سنرى فيما بعد أيضاً. شكل ٣-٩ يبين رسمياً تخطيطياً لهذا المسجل.

TCNT1L[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
TCNT1H[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

القيم التلقائية عند بداية التشغيل

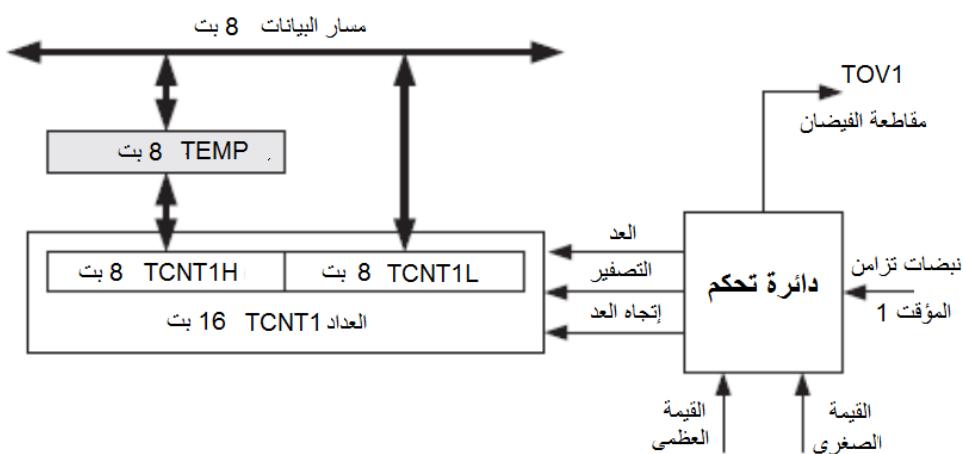
نلاحظ من الشكل ٣-٩ أن العداد يتكون من ١٦ بت توضع البایت ذات القيمة الأدنى في المسجل TCNT1L وهو مسجل ٨ بت توضع فيه قيمة البایت اليمني من العداد، كما توضع البایت ذات القيمة الأعلى في المسجل TCNT1H وهو مسجل ٨ بت أيضاً توضع فيه البایت ذات القيمة اليسري من العداد، والمسجلان يكونان مسجل عداد المؤقت ١ المكون من ١٦ بت، TCNT1.

عندما تتعامل وحدة المعالجة المركزية CPU في المتحكم مع هذا المسجل المكون من ١٦ بت (أو مع أي مسجل آخر ١٦ بت) سواء بالقراءة أو الكتابة فيه، في حين أن مسار البيانات مكون فقط من ٨ بت، فإن هذه العمليات تتم على خطوتين وبمساعدة مسجل مؤقت temporary register مكون من ٨ بت كما يلى:

- في حالة قراءة المسجل TCNT1 المكون من ١٦ بت، فإن النصف الأدنى منه TCNT1L يتم قراءته أولاً وتوضع على مسار البيانات لتنذهب إلى أي مكان، وفي نفس الوقت فإن النصف الأعلى من المسجل TCNT1H وهو TEMP تم قراءته ووضعه في مسجل مؤقت TEMP كما في شكل ٤-٩. نؤكد

هنا على أن قراءة النصف الأعلى تكون في نفس وقت قراءة النصف الأدنى لأن العداد يكون مستمراً في العد ويجب قراءة النصفين في نفس الوقت حتى يمثلا قيمة العداد عند نفس اللحظة. بعد قراءة المسجلين، فإن المسجل TCNT1L يوضع على مسار البيانات ليذهب إلى وجهته، والمسجل TCNT1H يوضع في المسجل TEMP لتتم قراءته في دورة القراءة التالية.

٢- في حالة الكتابة في المسجل TCNT1 المكون من ١٦ بت، فإن البایت ذات القيمة العظمى توضع أولاً في المسجل TEMP مع أول دورة كتابة. في دورة الكتابة التالية تتم كتابة البایت ذات القيمة الصغرى مباشرة في المسجل TCNT1L، حيث مع هذه العملية يتم في نفس الوقت كتابة محتويات المسجل TEMP في المسجل TCNT1H. وبذلك نضمن كتابة معلومة من ١٦ بت قادمة من مسار البيانات (٨ بت) على دوريٍّ كتابة في نفس الوقت في المسجلين TCNT1H و TCNT1L كما في شكل ٤-٩. فكرة القراءة أو الكتابة باستخدام المسجل TEMP كما سبق تتم مع أي مسجل ١٦ بت.



شكل ٤-٩ قراءة وكتابة العداد TCNT1 (١٦ بت) من خلال مسار بيانات ٨ بت  
(نسخة من دليل المتحكم atmeg328)

## ٤-٩ مسجل مقارنة الخرج A

يتكون مسجل مقارنة الخرج A للمؤقت ١ output compare register A, OCR1A من ١٦ بت أيضاً، ويتم فيه تسجيل القيمة التي سيتم مقارنتها مع محتويات مسجل المؤقت/العداد TCNT1 السابق بعد كل نبضة تزامن،

وعلى ضوء هذه المقارنة، وعند تساوى القيمتين، يتم إخراج إشارة معينة على طرف خرج المقارنة A رقم ١٥ للمتحكم وهو الطرف OC1A كما في شكل ١-٩، وسيأتي تفصيل لهذا الخرج فيما بعد. شكل ٥-٩ يبين رسمًا تخطيطيا لهذا المسجل. تذكر أنه يتم استخدام فكرة المسجل TEMP أيضا لضمان القراءة أو التسجيل في جزئي هذا المسجل كما أشرنا مسبقا.

7	6	5	4	3	2	1	0
OCR1AL[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
OCR1AH[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

القيم التلقائية عند بداية التشغيل

شكل ٥-٩ مسجل مقارنة الخرج A للمؤقت ١ OCR1A

## B-٥ مسجل مقارنة الخرج

يتكون مسجل مقارنة الخرج B OCR1B للمؤقت ١ output compare register B من ١٦ بت أيضا، ويتم فيه تسجيل القيمة التي سيتم مقارنتها مع محتويات مسجل المؤقت/العداد TCNT1 السابق بعد كل نبضة تزامن، وعلى ضوء هذه المقارنة، وعند تساوى القيمتين، يتم إخراج إشارة معينة على طرف خرج المقارنة B رقم ١٦ للمتحكم وهو الطرف OC1 كما في شكل ١-٩، وسيأتي تفصيل لهذا الخرج فيما بعد. شكل ٦-٩ يبين رسمًا تخطيطيا لهذا المسجل. تذكر أنه يتم استخدام فكرة المسجل TEMP أيضا لضمان القراءة أو التسجيل في جزئي هذا المسجل كما أشرنا مسبقا.

7	6	5	4	3	2	1	0
OCR1BL[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
OCR1BH[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

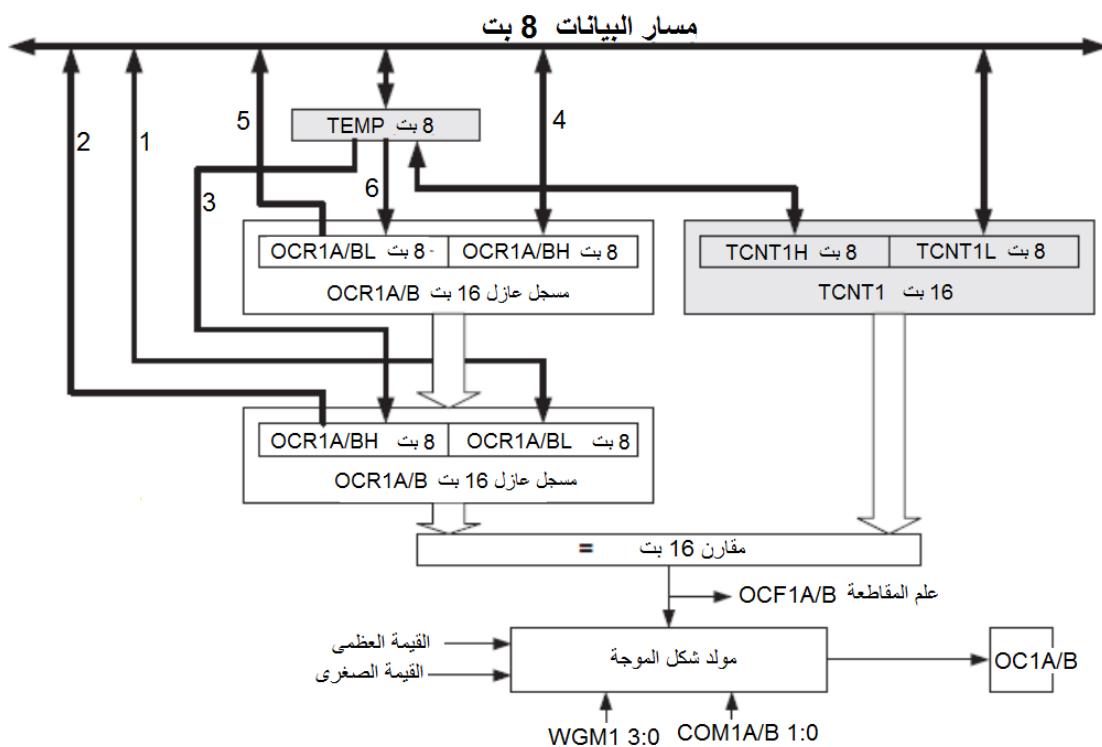
القيم التلقائية عند بداية التشغيل

شكل ٦-٩ مسجل مقارنة الخرج B للمؤقت ١ OCR1B

مثلاً كان الحال مع المؤقت صفر، فإن المؤقت ١ سيقارن قيمة العداد TCNT1 باستمرار مع قيمة مسجل المقارنة ISR OCR1A/B وعنده التساوى سيُضِع علم المقارنة OCF1A/B بواحد تمهيداً للقفز إلى برنامج خدمة مقاطعة إذا كان قناع هذه المقاطعة بواحد وعلم المقاطعة العام I بواحد. عند القفز إلى برنامج خدمة المقاطعة فإن علم المقاطعة يرجع للصفر مرة ثانية تمهيداً لمقارنة تالية. مولد شكل الموجة يستخدم هذا التساوى بين قيمة العداد ومسجلات المقارنة في إخراج شكل موجى يتوقف على حالة البتات 3:0 WGM ٧-٩ كما في شكل ٧-٩، وكما سيأتي شرحه بالتفصيل فيما بعد.

لاحظ في شكل ٧-٩ وجود مسجلين عازلين يتم تنشيطهما أو عدم تنشيطهما على حسب الحالة mode الذي يعمل فيه المؤقت ١ عند التعامل مع مسجلات المقارنة OCR1A/B. هناك بعض الحالات تحتاج لتنشيط العازلين معاً، وبعض الحالات لا تستدعى ذلك. يتم استخدام العازلين من أجل التزامن مع حواف نبضات التزامن الخاصة بالمؤقت وعدم حدوث نبضات دبوسية glitches في موجة الخرج. في الحالات التي لا تحتاج العازلين، فإن التعامل يكون مباشرةً بين مسار البيانات ومسجل المقارنة. القراءة من مسجل المقارنة في هذه الحالات لا تحتاج للمسجل OCR1A/BL، كما هو موضح في شكل ٧-٩ بالسهمين ١ و ٢. في هذه الحالة يتم قراءة البایت اليمى TEMP ثم البایت اليسرى OCR1A/BH دون الحاجة إلى المسجل TEMP، لأن عملية قراءة مسجل المقارنة لن تؤثر على تتبع عملية المقارنة. أما عملية الكتابة في الحالات التي لا تحتاج للمسجلين العازلين فإنها تتم مباشرةً أيضاً مع مسجل المقارنة OCR1A/B ولكن من خلال المسجل TEMP في هذه الحالة كما هو موضح بالسهمين ١ و ٣. في هذه الحالة تتم كتابة البایت اليسرى أولاً في المسجل TEMP، ثم بعد ذلك تتم كتابة البایت اليمى في المسجل OCR1A/BL مباشرةً حيث معها وفي نفس الوقت تنتقل البایت اليسرى من المسجل TEMP إلى المسجل OCR1A/BH. وبذلك نضمن أن كل من البایت اليمى واليسرى سيتم كتابتهما في نفس الوقت في مسجل المقارنة OCR1A/B. لاحظ أن عملية الكتابة في مسجل المقارنة تؤثر على تتبع المقارنة، لأن الكتابة تغير من محتويات هذه المسجلات على عكس عملية القراءة.

في الحالات التي تحتاج للمسجلين العازلين، فإن عملية القراءة ستكون مباشرةً من خلال المسجلين العازلين ولا حاجة هنا للمسجل TEMP كما بين ذلك السهمين ٤ و ٥. أما في حالة الكتابة في مسجل المقارنة في هذه الحالة فإنها ستتم من خلال المسجل TEMP وسيكون التعامل في هذه الحالة مع المسجلين العازلين أيضاً كما بين ذلك السهمين ٤ و ٦ في شكل ٧-٩. لاحظ في هذا الشكل أن نفس المسجل TEMP يستخدم أيضاً في حالة القراءة والكتابة من المسجل TCNT1 كما ذكرنا مسبقاً.



شكل ٧-٩ رسم تخطيطي لعملية المقارنة في المؤقت ١ (نسخة من دليل المتحكم atmega328)

## ٦-٩ مسجل مسک الدخل

مسجل مسک الدخل ICR1 هو مسجل من ١٦ بت يتكون من ٢ بايت، البايت اليمني وهي المسجل ICR1L، والبايت اليسرى وهي المسجل ICR1H الموضحان في شكل ٨-٩. حيث أن هذا المسجل مكون من ١٦ بت، فإن التعامل بينه وبين وحدة المعالجة المركزية CPU سيكون من خلال المسجل المؤقت TEMP وبنفس الطريقة التي سبق شرحها مع العداد INTC1.

يستخدم المؤقت ١ وحدة مسک الدخل capture the input حيث تقوم هذه الوحدة بمسک أحداث معينة على الدخل على خط معين وهو الخط ICP1 (رقم ١٤ في شريحة المتحكم)، أو دخل المقارن التماثلي، وتحديد زمن حدوث هذا الحدث بالضبط. بذلك يمكن تحديد الزمن بين حدثين، أو معدل حدوث حدث معين، أو خواص أخرى للإشارة الموجودة على الطرف ICP1. شكل ٩-٩ يبين رسمًا تخطيطياً لهذه الوحدة.

الأحداث الخارجية يمكن أن تكون في صورة أي تغير منطقي على الطرف ICP1، أو تغير في خرج المقارن التماثلي، ACO، الذي سيأتي شرحه فيما بعد. كما في شكل ٩-٩ فإن متعدد المدخلات MUX يختار واحد من هذين

الدخلين عن طريق قيمة البت ACIC، وهذا الدخل المختار يمر على بلوك خاص بالخلص من الضوضاء noise canceller ثم بلوك خاص بتحديد حافة هذا التغيير وجعلها تتوافق مع نبضات التزامن. هذا الحدث، بعد تحديد حافته، سيجعل علم المقاطعة الخاص بمسك الدخل ICF1 يساوى واحد كما في الشكل. بذلك يمكن القفز إلى برنامج خدمة مقاطعة ISR خاص بذلك لعمل أي فعل يريده المستخدم عند حدوث هذا الحدث. بالطبع فإن العلم ICF1 سيعود للصفر بمجرد القفز إلى برنامج خدمة المقاطعة تلقائيا حتى يكون المتحكم جاهز لاستقبال أي حدث آخر.

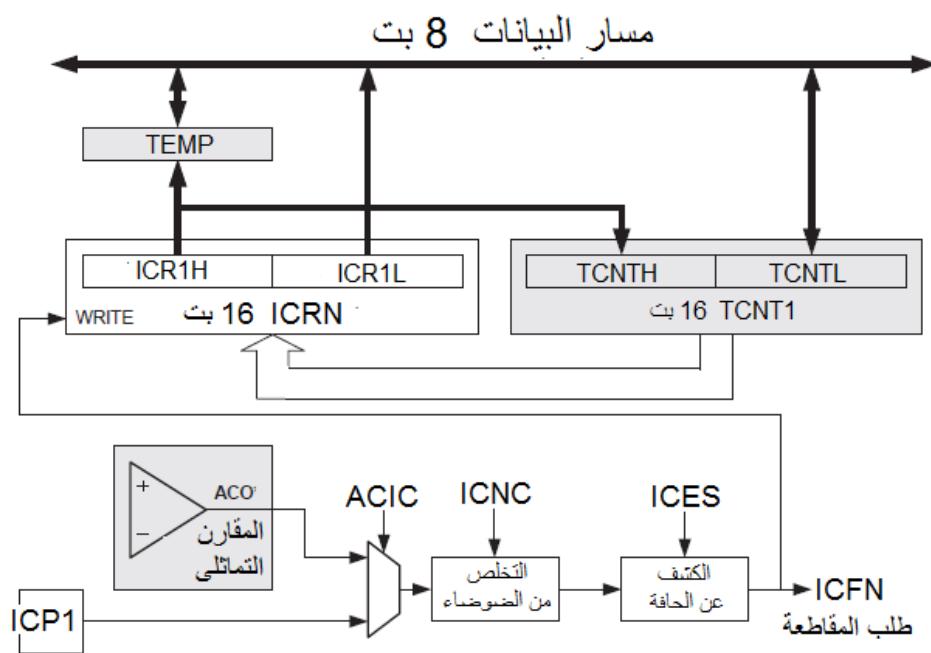
7	6	5	4	3	2	1	0
ICR1L[7:0]							
R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0
7	6	5	4	3	2	1	0
ICR1H[15:8]							
R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0

القيم التلقائية عند بداية التشغيل

شكل ٨-٩ مسجل مسک الدخل في المؤقت ١

هذه الحافة الناتجة من الحدث، ستعمل على تنشيط مسجل مسک الدخل ICR1 لتجعله جاهز للكتابة من خلال طرف الكتابة write، حيث على الفور يتم تسجيل قيمة العداد TCNT1 في المسجل ICR1 وبالطبع فإن هذه القيمة تمثل زمن حدوث الحدث. لاحظ أن كلا المسجلين ICR1 و TCNT1 مكون من ١٦ بت، ولذلك فعملية الكتابة من الأول للثاني ستتم مباشرة. عملية قراءة مسجل مسک الدخل ICR1 من خلال مسار البيانات ستتم بالطبع من خلال المسجل TEMP بالطريقة المعتادة التي ذكرناها عند تعامل مسار البيانات (٨ بت) مع أي مسجل ١٦ بت.

البلوك الخاص بالخلص من الضوضاء يتم تنشيطة على حسب الحاجة بتنشيط البت ICNC وهي أحد برات مسجل تحكم المؤقت ١ Timer Counter Control Register، TCCR1 التي سيأتي شرحها بالتفصيل بعد قليل. هذا البلوك يعمل بمثابة مرشح رقمي يقوم بمقارنة قيمة الإشارة الداخلية على مدار أربع نبضات تتوافق من نبضات المتحكم نفسه وليس من نبضات المؤقت (بعد القسمة على معامل القسمة)، فإذا كانت الإشارة مستقرة ثابتة على مدار الأربع نبضات تقريبا فإنها تمر على أنها إشارة محققة، وإذا حدث تغيير في إشارة الدخل أثناء هذه النبضات الأربع، فإنه يتم إهمال هذه العينة من الإشارة واعتبارها ضوضاء. تؤكد هنا على نبضات التزامن التي يعمل عندها هذا البلوك هي نبضات المتحكم الأصلية، كما أنه يمكن تنشيط هذا البلوك أو عدم تنشيطة على حسب رغبة المستخدم.



شكل ٩-٩ مخطط تفصيلي لعملية مسک الدخل Input Capture (نسخة من دليل بيانات المتحكم atmega328)

## ٧-٩ مقاطعة المؤقت ١

المؤقت ١ له أربعة مصادر للمقاطعة كما هو مبين في الجدول ١-٩ الذي يوضح مصدر كل مقاطعة، والعنوان الذي يتم القفز إليه عند حدوث المقاطعة، ورقم متوجه المقاطعة. المقاطعة الأولى تقع عند حدوث حدث على أحد مصادر مسک الدخل Timer1 CAPT، والمصدر الثاني يحدث عند التساوى بين محتويات مسجل المؤقت ١ TCNT1 ومسجل مقارنة الخرج OCR1A. عند وقوع هذا التساوى بشرط أن يكون قناع المقاطعة الخاص بهذه المقاطعة نشطاً، وأن يكون علم المقاطعة العام في مسجل الأعلام نشطاً، فإن المتحكم سيقفز إلى العنوان 0x00B كما في الجدول ١-٩. المقاطعة الثالثة تقع عند حدوث تساوى بين محتويات مسجل المؤقت ١ TCNT1 ومسجل مقارنة الخرج OCR1B. عند وقوع هذا التساوى بشرط أن يكون قناع المقاطعة الخاص بهذه المقاطعة نشطاً، وأن يكون علم المقاطعة العام في مسجل الأعلام نشطاً، فإن المتحكم سيقفز إلى العنوان 0x00C كما في الجدول ١-٩. المقاطعة الرابعة تقع عند حدوث فيضان في مسجل المؤقت ١ TCNT1، بمعنى أن تصل محتوياته إلى ٦٥٥٣٦، حيث عندها يقفز المتحكم إلى

العنوان 0x00D يشرط أن يكون كل من قناع هذه المقاطعة وقناع المقاطعة العام في مسجل الأعلام نشطين، كما سنرى في المسجلات التالية.

جدول ١-٩ إسم ومصدر وعنوان ورقم متوجه المقاطعات في المؤقت ١ (نسخة من دليل المتحكم  
(atmeg328

Vector No.	Program Address	Source	Interrupt Definition
8	0x007	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x008	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x009	TIMER2 OVF	Timer/Counter2 Overflow
11	0x00A	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x00B	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x00C	TIMER1 COMPB	Timer/Counter1 Compare Match B
14	0x00D	TIMER1 OVF	Timer/Counter1 Overflow
15	0x00E	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x00F	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x010	TIMER0 OVF	Timer/Counter0 Overflow

### مسجل أقنعة مقاطعات المؤقت ١ Timer Interrupt Mask Register, TIMSK1

يبين شكل ١٠-٩ بذات هذا المسجل، حيث نلاحظ استخدام أربع برات (تقابل الأربع مقاطعات) من هذا المسجل وهي كالتالي:

**البت ٠: قناع مقاطعة الفيضان Timer Over flow Interrupt Enable, TOIE**، حيث بوضع واحد في هذه البت يتم تنشيط مقاطعة الفيضان ويقفر المتحكم إلى العنوان 0x00D إذا كان علم المقاطعة العام I يساوى واحد أيضاً.

**البت ١: قناع مقاطعة مقارنة الخرج Output Compare Interrupt Enable A, OCIEA**، حيث بوضع واحد في هذه البت يتم تنشيط مقاطعة تساوى العداد TCNT1 ومسجل مقارنة الخرج A ويقفر المتحكم إلى العنوان 0x00B إذا كان علم المقاطعة العام I يساوى واحد.

**البت 2: قناع مقاطعة مقارنة الخرج Output Compare Interrupt Enable B, OCIEB**, حيث يوضع واحد في هذه البت يتم تنشيط مقاطعة تساوى العداد TCNT1 ومسجل مقارنة الخرج B ويقفل المتحكم إلى العنوان 0x00C إذا كان علم المقاطعة العام I يساوى واحد.

**البت 5: قناع مقاطعة مسك الدخل Input Capture Interrupt Enable, ICIE**, يوضع واحد في هذه البت، وعندما يكون علم I أو قناع المقاطعة العام I نشطا، فإنه عند حدوث أى حدث مسك للدخل بحيث يصبح علم مسك الدخل ICF1 (في مسجل أعلام المقاطعة التالي) يقفل المتحكم إلى برنامج خدمة المقاطعة ISR الذي مؤشره في العنوان 0x00A.

7	6	5	4	3	2	1	0
0	0	ICIE 0	0		OCIEB 0	OCIEA 0	TOIE 0

القيم التلقائية عند بداية التشغيل

شكل ١٠-٩ مسجل أقنعة مقاطعات المؤقت ١

باقي بิตات هذا المسجل غير مستخدمة.

في كل المقاطعات الثلاث السابقة كيف يعرف المتحكم بحدوث تساوى بين مسجلات المقارنة A أو B مع مسجل المؤقت أو حدوث فيضان في محتويات العداد، أو مسك أحد أحداث الدخل. يتم ذلك عن طريق علم مقاطعة خاص بكل منها في المسجل التالي.

### مسجل أعلام مقاطعة المؤقت ١ Timer/Counter1 Interrupt Flag Register, TIFR1

يبين شكل ١١-٩ بิตات هذا المسجل، حيث نلاحظ استخدام أربع بิตات فقط من هذا المسجل وهي كالتالي:

**البت ٠: علم حدوث الفيضان Timer Over flow Flag, TOV**, هذه البت (العلم) تصبح واحد عند حدوث فيضان في محتويات عدد المؤقت TCNT1، ويتم تصفيرها عن طريق المتحكم بمجرد القفز إلى برنامج خدمة المقاطعة ISR الخاص بها.

**البت 1: علم مقارنة الخرج Output Compare A Interrupt Flag, OCFA**, هذه البت (العلم) تصبح واحد عند تساوى العداد TCNT1 ومسجل مقارنة الخرج A، ويتم تصفيرها عن طريق المتحكم بمجرد القفز إلى برنامج خدمة المقاطعة ISR الخاص بها.

**البت 2: علم مقارنة الخرج Output Compare B Interrupt Flag, OCFB**, هذه البت (العلم) تصبح واحد عند تساوى العداد TCNT1 ومسجل مقارنة الخرج B، ويتم تصفيرها عن طريق المتحكم بمجرد القفز إلى برنامج خدمة المقاطعة ISR الخاص بها.

**البت 5: علم مسك الدخل Input Capture Interrupt Flag, ICF**, هذا العلم يصبح واحد عند حدوث مسک للدخل على الطرف ICP1 لشريحة المتحكم. بمجرد القفز إلى برنامج خدمة المقاطعة ISR يتم تصفير هذا العلم.

باقي بิตات هذا المسجل غير مستخدمة.

7	6	5	4	3	2	1	0
0	0	0	0		OCFB	OCFA	TOV

القيم التلقائية عند بداية التشغيل

شكل ١١-٩ مسجل أعلام مقاطعات المؤقت ١

إذن باختصار يمكننا أن نلخص آلية المقاطعات الأربع السابقة بأنه بمجرد حدوث تساوى بين مسجل عداد المؤقت ١ TCNT1 وأحد مسجلات المقارنة A أو B، أو حدوث فيضان في محتويات المسجل OCFB، أو حدوث مسک للدخل، فإن العلم المقابل لكل منهما TOV أو OCFA أو OCFB، أو ICF يصبح واحد، وبالتالي إذا كان قناع المقاطعة المقابل لكل واحدة من هذه المقاطعات TOIE أو OCIEA أو ICIE يساوى واحد، وإذا كان علم المقاطعة العام I في مسجل الحاله يساوى واحد أيضاً، فإن المقاطعة ستحدث، ويقفز المتحكم إلى برنامج خدمة المقاطعة ISR المقابل لأى منهم، وبمجرد انتقال المتحكم إلى برنامج خدمة المقاطعة ISR فإن العلم المقابل يتم تصفيره مرة ثانية استعداداً لحدث مقاطعة أخرى.

## ٨-٩ مسجلات التحكم في أداء المؤقت ١

جدول ٢-٩ وصف بثات حالات تشغيل المؤقت ١ (نسخة من دليل المتحكم ٣٢٨ (atmega328

الحالة Mode	WGM13	WGM12	WGM11	WGM10	حالات التشغيل	القيمة العظمى	تجدد قيمة مسجل المقارنة	لحظة وضع علم الفيضان بواحد
0	0	0	0	0	العادى	0xFF	فورى	MAX
1	0	0	0	1	موجة معدلة العرض، تصحيح الطور بت ٨	0x00FF	TOP	Bottom
2	0	0	1	0	موجة معدلة العرض، تصحيح الطور بت ٩	0x01FF	TOP	Bottom
3	0	0	1	1	موجة معدلة العرض، تصحيح الطور بت ١٠	0x03FF	TOP	Bottom
4	0	1	0	0	CTC	OCR1A	immediate	MAX
5	0	1	0	1	Fast PWM, 8 bit	0x00FF	Bottom	TOP
6	0	1	1	0	Fast PWM, 9 bit	0x01FF	Bottom	TOP
7	0	1	1	1	Fast PWM, 10 bit	0x03FF	Bottom	TOP
8	1	0	0	0	موجة معدلة العرض، تصحيح طور وتردد	ICR1	Bottom	Bottom
9	1	0	0	1	موجة معدلة العرض، تصحيح طور وتردد	OCR1A	Bottom	Bottom
10	1	0	1	0	موجة معدلة العرض، تصحيح طور	ICR1	TOP	Bottom
11	1	0	1	1	موجة معدلة العرض، تصحيح طور	OCR1A	TOP	Bottom
12	1	1	0	0	CTC	ICR1	immediate	MAX
13	1	1	0	1	غير مستخدم	-----	-----	-----
14	1	1	1	0	Fast PWM	ICR1	Bottom	TOP
15	1	1	1	1	Fast PWM	OCR1A	Bottom	TOP

يتم التحكم في أداء المؤقت ١ عن طريق ثلاثة مسجلات A و B و C لهذا الغرض، وسنعرض بعده كل من هذه المسجلات بالتفصيل في هذا الجزء.

### مسجل التحكم A في المؤقت ١ Timer/Counter1 Control Register, TCCR1A

البتات ٠ و ١ في مسجل التحكم **TCCR1A** والبتات ٣ و ٤ في مسجل التحكم **TCCR1B**، وهي: WGM13 و WGM12 و WGM11 و WGM10: هذه البتات الأربع خاصة بالتحكم في تتابع عملية العد في العدد، وفي مصدر القيمة العظمى التي يمكن أن يصل إليها العدد، وفي نوع الموجة المولدة على طرف خرج المقارنة التي يمكن استخدامها، حيث يمكن استخدام واحد من ستة عشر طريقة أو ستة عشر حالة للتشغيل أو للحصول على موجة بشكل معين مثل الموجة المربعة المعدلة العرض PWM كما سنرى بالتفصيل فيما بعد. جدول ٢-٩ يبين هذه الحالات الستة عشرة للتشغيل في مقابل قيمة كل بت من هذه البتات. شكل ١-٩ يبيّن مخطط لبتات هذا المسجل. سينتني شرح هذه الحالات بالتفصيل بعد قليل.

7	6	5	4	3	2	1	0
COM1A1	COM1A0	COM1B1	COM1B0	0		WGM11	WGM10

القيم التلقائية عند بداية التشغيل

شكل ١-٩ المسجل A للتحكم في المؤقت ١

جدول ٣-٩ حالات خرج المقارنة مع مسجل المقارنة A وفي غير حالة التشغيل PWM للمؤقت ١

وصف الحالة	
الطرف OC1A يعمل كطرف بوابة عادي، ليس له علاقة بالمقارنة	Normal port operation
عند حدوث التساوى يغير الخط OC1A من حالته، فإذا كان صفر يصبح واحد، وإذا واحد يصبح صفر.	Toggle on compare match
عند حدوث التساوى يصبح الطرف OC1A صفر، clear on compare match	
عند حدوث التساوى يصبح الطرف OC1A واحد، Set on compare match	

**البنا ٦ و ٧ في المسجل TCCR0A:** هذه البنا COM1A0 و COM1A1 تتحكم في سلوك الخرج على طرف تساوى المقارنة OC1A، من حيث هل هذا الخط سيغير حالته مع كل تساوى بين مسجل المقارنة OCRA و المسجل TCNT1 أم أنه سيصبح صفر أم سيكون واحد. كل هذه الحالات بيبيها الجدول ٣-٩. لاحظ أن الطرف OC1A وهو الطرف ١٥ في شريحة المتحكم لابد أن يتم تشغيله كطرف خرج من خلال مسجل الاتجاه PDDR الخاص بالبوابة التابع لها هذا الطرف وهي البوابة B في هذه الحالة. الحالات الموجودة في الجدول ٣-٩ للطرف OC1A تكون حالات التشغيل التي لا يكون فيها تعديل موجي على عرض الموجة الخارجية PWM حيث في هذه الحالة ستعمل هذه البنا بطريقة أخرى سنشرحها بعد قليل مع شرح حالات التشغيل.

**البنا ٤ و ٥ في المسجل TCCR1A:** هذه البنا COM1B0 و COM1B1 تتحكم في سلوك الخرج على طرف تساوى المقارنة OC1B، من حيث هل هذا الخط سيغير حالته مع كل تساوى بين مسجل المقارنة OCRB و المسجل TCNT1 أم أنه سيصبح صفر أم سيكون واحد. كل هذه الحالات بيبيها الجدول ٤-٩. لاحظ أن الطرف OC1B وهو الطرف ١٦ في شريحة المتحكم لابد أن يتم تشغيله كطرف خرج من خلال مسجل الاتجاه PDDR الخاص بالبوابة التابع لها هذا الطرف وهي البوابة B في هذه الحالة. الحالات الموجودة في الجدول ٤-٩ للطرف OC1B تكون حالات التشغيل التي لا يكون فيها تعديل موجي على عرض الموجة الخارجية PWM حيث في هذه الحالة ستعمل هذه البنا بطريقة أخرى سنشرحها بعد قليل مع شرح حالات التشغيل.

**البنا ٢ و ٣ في المسجل TCCR1A غير مستخدمة.**

جدول ٤-٩ حالات خرج المقارنة مع مسجل المقارنة B وفي غير حالة التشغيل PWM للمؤقت ١

(نسخة من دليل المؤقت ٣٢٨ atmega328)

وصف الحالة	COM1B1	COM1B0
الطرف OC1B يعمل كطرف بوابة عادي، ليس له علاقة بالمقارنة Normal port operation	0	0
عند حدوث التساوى يغير الخط OC1B من حالته، فإذا كان صفر يصبح واحد، وإذا واحد يصبح صفر. Toggle on compare match	0	1
عند حدوث التساوى يصبح الطرف OC1B صفر، Clear on compare match	1	0
عند حدوث التساوى يصبح الطرف OC1B واحد، Set on compare match	1	1

**مسجل التحكم B في المؤقت ١ Timer/Counter1 Control Register, TCCR1B**

7	6	5	4	3	2	1	0
ICNC1	ICES1		WGM13	WGM12	CS12	SC11	CS10
0	0		0	0	0	0	0

القيم التلقائية عند بداية التشغيل

شكل ١٣-٩ المسجل B للتحكم في المؤقت ١

شكل ١٣-٩ يبين رسمياً تخطيطياً لبيانات المسجل B للتحكم في أداء المؤقت ١، وتفاصيل هذه البيانات ستكون كما يلى:

**البيانات ٠ و ١ و ٢ في المسجل TCCR1B:** هذه البيانات CS10 و CS11 و CS12 يتم عن طريقها اختيار مصدر نبضات تزامن المؤقت/العداد ١ تبعاً للجدول ٥-٩.

جدول ٥-٩ اختيار مصدر نبضات التزامن للمؤقت/العداد ١ (أنظر شكل ٢) (نسخة من دليل المتحكم atmeg328)

الوصف
لا يوجد مصدر، ويتوقف المؤقت No clock source
نفس نبضات المتحكم، معامل قسمة يساوى ١ ، No prescaling
نبضات المتحكم على ٨
نبضات المتحكم على ٦٤
نبضات المتحكم على ٢٥٦
نبضات المتحكم على ١٠٢٤
مصدر النبضات هو الطرف T0 ويعمل على الحافة النازلة
مصدر النبضات هو الطرف T0 ويعمل على الحافة الصاعدة

**البيانات ٣ و ٤ في المسجل TCCR1B:** هذه البيانات هى WGM12 و WGM13 وهى كما ذكرنا من قبل خاصة باختيار شكل موجة الخرج على الطرف OC1A/B، أو بمعنى آخر اختيار طريقة من ١٦ طريقة تشغيل للمؤقت ١ بالتعاون مع البيانات WGM11 و WGM10 في مسجل التحكم TCCR1A.

**البيت ٦ في المسجل TCCR1B:** هذه البيط هي ICES1 وهى تختار الحافة على خط مسک الدخل ICP1 التي ستحدث عندها عملية المسك للدخل. بوضع هذه البيط تساوى واحد، فإن الحافة الصاعدة على هذا الدخل ستحدث عملية المسك، وبوضع هذه البيط بصفر فإن عملية المسك ستحدث مع الحافة النازلة على الطرف ICP1. وقد

أوضحنا أن عملية المسك تعنى أن محتويات العداد TCNT1 سيتم مسكتها في مسجل المسك ICR1 عند هذه اللحظة، وسيتم بناء على ذلك تنشيط علم المسك ICF1 يجعله يساوى واحد تمهيدا للقفز إلى برنامج خدمة المقاطعة ISR. انظر شكل ٩-٩.

**البت 7 في المسجل TCCR1B:** هذه البت هي ICNC1، بوضع هذه البت تساوى واحد يتم تنشيط بلوك التخلص من الضوابط التي على طرف المسك ICP1 كما في شكل ٩-٩ وكما أشرنا مسبقا. وبالطبع بوضعها تساوى صفر فلن يتم تنشيط هذا البlok.

### مسجل التحكم C في المؤقت/العداد ١ Timer/Counter1 Control Register, TCCR1C

هذا المسجل مستخدم منه آخر اثنين بت فقط وهما البت FOC1A و FOC1B كما في شكل ١٤-٩.

7	6	5	4	3	2	1	0
FOC1A	FOC1B						

القيم التلقائية عند بداية التشغيل  
0      0

شكل ١٤-٩ المسجل C للتحكم في المؤقت ١

**البت 7 في المسجل TCCR1C:** هذه البت FOC1A، عندما تكون واحد تسبب في إحداث مقارنة فورية بين مسجل المقارنة OCRA و المسجل TCNT1 والتأثير على طرف خرج المقارنة OC1A تبعا للأطراف COM1A1 و COM1A0 الموضحة في جدول ٣-٩. هذه البت لا تعمل إلا مع حالات التشغيل التي لا يكون فيها تعديل لعرض الموجة PWM، أى أن يكون معطلا عن طريق بتات شكل موجة التعديل WGM، لذلك يجب أن تكون هذه البت صفر في حالة العمل في هذه الحالات من التشغيل.

**البت 6 في المسجل TCCR1C:** هذه البت FOC1B، عندما تكون واحد تسبب في إحداث مقارنة فورية بين مسجل المقارنة OCRB و المسجل TCNT1 والتأثير على طرف خرج المقارنة OC1B تبعا للأطراف COM1B1 و COM1B0 الموضحة في جدول ٤-٩. هذه البت لا تعمل إلا مع حالات التشغيل التي لا يكون فيها تعديل لعرض الموجة PWM، أى أن يكون معطلا عن طريق بتات شكل موجة التعديل WGM، لذلك يجب أن تكون هذه البت صفر في حالة العمل في هذه الحالات من التشغيل.

## ٩-٩ حالات تشغيل المؤقت ١

تتحدد حالات التشغيل، أو أداء المؤقت على أطراف المقارنة OC1A أو OC1B على ضوء بثات اختيار الشكل الموجى على الأطراف WGM في المسجلين TCCR1A و TCCR1B، والبتابات COM1A/B0 و COM1A/B1 في المسجل TCCR1A. هناك ١٦ من هذه الحالات التي سنقدمها في هذا الجزء والتي تم عرضها في الجدول ٢-٩.

### ١- حالة التشغيل العادي Normal mode

هذه الحالة من حالات التشغيل هي أبسط حالة في الحالات الستة عشرة. يتم الدخول في هذه الحالة بوضع بثات شكل الموجة WGM10 و WGM11 و WGM12 و WGM13 كلها بأصفار كما في الجدول ٢-٩. في هذه الحالة يقوم العداد TCNT1 بالعد دائمًا في الاتجاه التصاعدي بزيادة مقدارها واحد، ولا يتم تصفير العداد نهائياً إلا عند وصوله للقيمة العظمى TOP=0xFFFF، حيث عندها يبدأ العداد في العد مرة ثانية من القيمة الصغرى Bottom=0x0000. بمجرد أن تصبح قيمة العداد TCNT1 تساوى صفر ومع نفس نبضة التزامن فإن علم الفيضان TOV1 يصبح واحد (أى أن هذه البت تسلك مسلك البت السابعة عشرة للعداد)، تمهدًا لقفز إلى برنامج خدمة المقاطعة ISR إذا كان قناع هذه المقاطعة وعلم المقاطعة العام نشطين كما أشرنا مسبقًا. هذا العلم يتم تصفيره بمجرد القفز إلى برنامج خدمة المقاطعة ISR تمهدًا لدورة العد التالية.

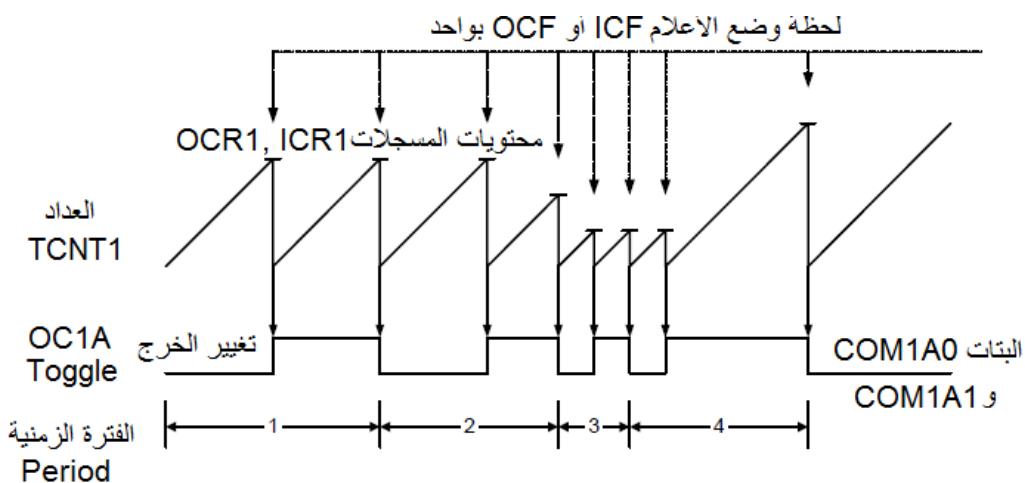
أسهل استخدام لوحدة مسک الدخل يكون مع هذه الطريقة للتشغيل العادي، مع ملاحظة أن يكون الزمن بين أي حدثين متاليين أقل من فترة العد للعداد من القيمة الصغرى للعظمى (تحديد resolution للعداد). بالطبع يمكن اللعب بمعامل القسمة لنبيضات تزامن المؤقت لتتمديد فترة العد ل تستوعب الفترة بين أي حدثين.

كمثال على التشغيل العادي يمكن تنفيذ نفس برنامج التشغيل العادي في الفصل ٨ مع مراعاة الفروق بين قيم بثات المسجلات في المؤقتين.

### ٢- تصفير المؤقت ١ عند تساوى المقارنة Clear Timer on Compare, CTC

كما في الجدول ٢-٩ فإن حالة تصفير المؤقت ١ عند تساوى المقارنة يمكن الحصول عليها من الحالة ٤ والحالة ١٢. في الحالة ٤ يتم تصفير العداد عند تساوى قيمة العداد TCNT1 مع قيمة مسجل مقارنة الخرج OCR1A، وفي الحالة ١٢ يتم تصفير العداد TCNT1 عند تساوى قيمته مع مسجل مسک الدخل ICR1، وبالتالي فإن المسجلين

و ICR1 OCR1 يحددان التحديدية resolution للعداد في هاتين الحالتين. شكل ١٥-٩ يبين مخطط التزامن لهذه الطريقة حيث نلاحظ زيادة قيمة العداد من الصفر إلى القيمة العظمى المحددة في أحد المسجلين OCR1 أو ICR1 ثم يرجع مرة ثانية إلى الصفر.

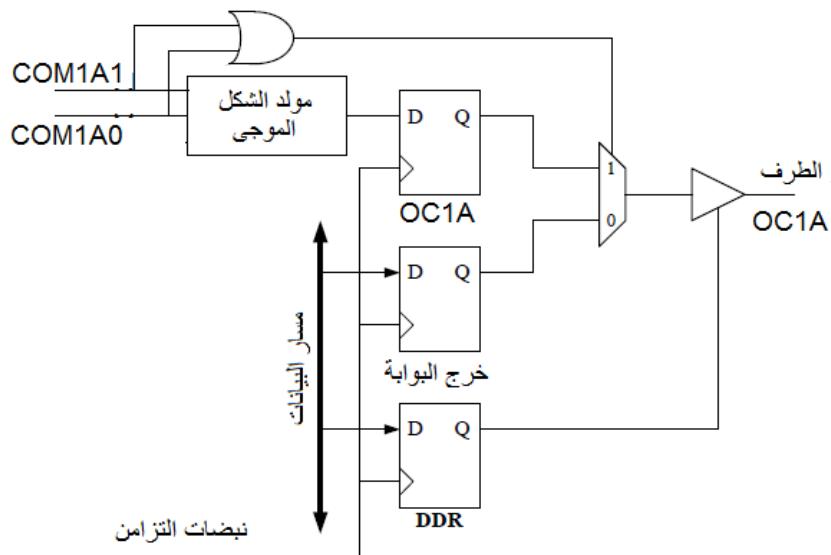


شكل ١٥-٩ المخطط الزمني لطريقة تشغيل تصفير المؤقت ١ عند تساوى المقارنة (نسخة من دليل المتحكم (atmega328

بمجرد تساوى قيمة العداد TCNT1 مع قيمة المسجل OCR1 أو ICR1 فإنه يحدث تنشيط لعلم المقاطعة الخاص بكل منها، وبالتالي فإذا كان قناع هذه المقاطعة نشطاً، وكان علم المقاطعة العام I نشطاً هو الآخر فسيتم القفز إلى برنامج خدمة المقاطعة ISR الخاص بكل منها. بمجرد الدخول في برنامج خدمة المقاطعة يتم تصفير هذه الأعلام مرة ثانية استعداداً لدورة جديدة. يمكن تغيير قيمة المسجلات OCR1 أو ICR1 في برنامج خدمة المقاطعة للحصول على أشكال موجية مختلفة (مختلفة الفترة الزمنية) كما في شكل ١٥-٩.

للحصول على أشكال موجية في الخرج مثل الموضحة في شكل ١٥-٩ على الطرف OC1A/B يتم وضع البتات COM1A1=1 و COM1A0=1 لكي يحدث تغيير للخرج عند كل تساوى للمقارنة. هذا الشكل الموجى لن يكون ظاهراً على الطرف OC1A/B إلا إذا كان هذا الطرف قد تم تعينه كطرف خرج في البوابة التي تحتوى على هذا الطرف وهى البوابة B مع المؤقت ١ تساوى واحد (DDRB1=1). شكل ١٦-٩ يوضح ذلك حيث أن بت اتجاه البوابة DDR هي التي تنشط العازل الثلاثي المنطق الموضع قبل طرف الخرج OC1A. هذا الشكل قد تم شرحه بالتفصيل مع المؤقت صفر في معرض الحديث عن نفس هذا الموضع. لذلك كان التأكيد فيما سبق على أنه لكي

يؤدي طرف المقارنة OC1A أو OC1B وظيفته، فإن طرف البوابة المقابل يجب أن يتم تعينه على أنه طرف خرج بوضع واحد في قلاب الاتجاه المناظر.



شكل ١٦-٩ علاقة طرف المقارنة OC1A مع طرف البوابة المقابل له

(نسخة من دليل المتحكم atmega328)

يمكن تحديد تردد الموجة الناشئة على الطرف OC0A تبعاً للمعادلة التالية:

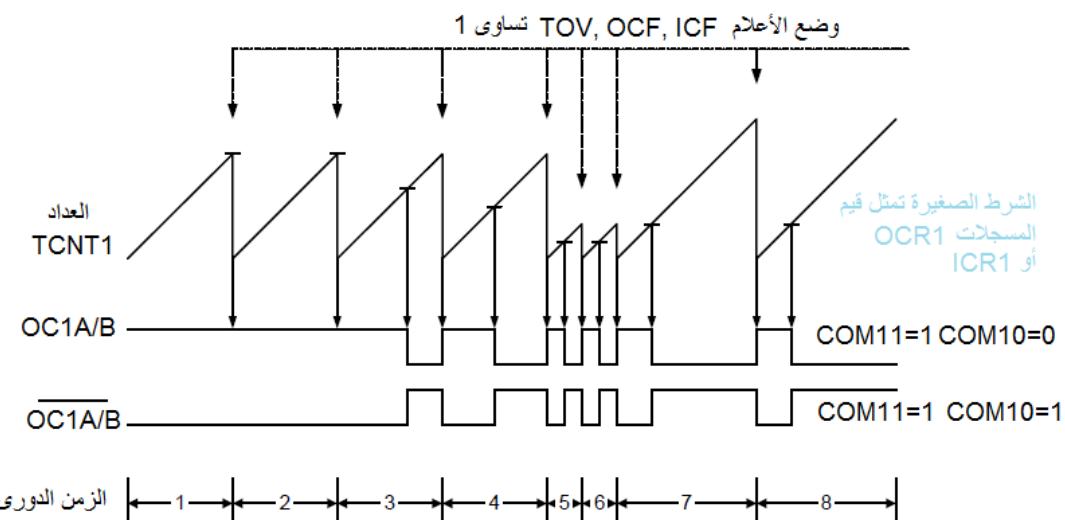
$$f_{OC1A} = \frac{f_{clock}}{2N(1+OCR1A)}$$

حيث N هي معامل القسمة (١ أو ٨ أو ٦٤ أو ٢٥٦) المستخدم للحصول على نبضات تزامن المؤقت،  $f_{clock}$  هي نبضات تزامن المتحكم. لاحظ أن أكبر تردد يمكن الحصول عليه في هذه الحالة سيكون عندما  $OCR0A=0$ . يمكن تنفيذ نفس المثال الخاص بهذه الطريقة الذي تم تقديمها مع المؤقت صفر مع ملاحظة الفروق في بثات الشكل الموجي التي في الجدول ٢-٩.

### ٣- طريقة تعديل عرض النبضة PWM السريعة

هذه الطريقة من التشغيل يمكن الحصول عليها بوضع بثات الشكل الموجي WGM10 و WGM11 و WGM12 و WGM13 تساوى ٥ أو ٦ أو ٧ أو ١٤ أو ١٥ كما في الجدول ٢-٩، حيث تختلف هذه الحالات باختلاف القيمة العظمى التي يصل إليها العداد ومصدرها، وعدد بثات العداد المستخدمة. في هذه الطريقة يبدأ العداد في العد

من الصفر إلى القيمة العظمى، ثم الصفر حتى القيمة العظمى، وهكذا. لذلك فإن هذه الطريقة تعرف بالليل الواحد. في حالة عدم العكس يتم تصفيير طرف خرج المقارنة OC1A/B عند حدوث التساوى بين قيمة العداد TCNT1 والقيمة المخزنة في مسجل المقارنة OCR1A/B، ويرجع للواحد عندما يصل العداد إلى القيمة العظمى وينزل بعدها للصفر. في حالة العكس يحدث عكس ذلك، كما في شكل ١٧-٩. في حالة التشغيل ٥ كما في جدول ٢-٩ يكون عدد بتات العداد يساوى ٨ بت وبالتالي فإن القيمة العظمى للعداد ستكون  $d_{TOP}=0x00FF=256$ ، وفي الحالة ٦ يكون عدد بتات العداد يساوى ٩ بت وبالتالي ستكون القيمة العظمى  $d_{TOP}=0x01FF=512$ . في الحالة ٧ يكون عدد بتات العداد يساوى ١٠ بت، وبالتالي تكون القيمة العظمى  $d_{TOP}=0x03FF=1024$ . في الحالة ١٤ تكون القيمة العظمى مخزنة في مسجل مسک الدخل ICR1 وفي الحالة ١٥ تكون القيمة العظمى مخزنة في مسجل خرج المقارنة OCR1A/B.



شكل ١٧-٩ التزامن الحادث مع طريقة تعديل عرض النبضة PWM السريع في حالة المؤقت ١  
(نسخة من دليل المتحكم atmega328)

إذن معنى ذلك أن الزمن الدورى للموجة الناتجة على الطرف OC1A سيكون ثابت، وأما نسبة الزمن ON للزمن OFF أثناء الزمن الدورى فسيتم التحكم فيها عن طريق القيمة الموجودة في مسجل المقارنة، ومن هنا كانت فكرة تعديل عرض الموجة PWM الموضحة في شكل ١٧-٩. في الحالة العاكسة يقل الزمن ON بزيادة القيمة المخزنة في المسجل OCR1A، وفي الحالة غير العاكسة، فإن الزمن ON يزيد بزيادة القيمة المخزنة في مسجل المقارنة OCR1A. تذكر هنا أنه لكي تتم رؤية هذه الموجات على طرف المقارنة OC1A، فإن هذا الطرف لابد أن يكون قد تم تعينه

ليكون طرف خرج بوضع وحيد في مسجل اتجاه البوابة التابع لها هذا الطرف كما في شكل ١٦-٩ ، وهي البوابة B في حالة المؤقت ١ .

تردد الموجة المعدلة العرض PWM السريع يمكن الحصول عليها من المعادلة التالية:

$$f_{OC0APWM} = \frac{f_{clock}}{N.(1 + TOP)}$$

حيث  $f_{clock}$  هي نبضات تزامن المتحكم، و N هي نسبة القسمة المستخدمة للحصول على نبضات التزامن التي سيعمل عندها المؤقت (١ أو ٨ أو ٦٤ أو ٢٥٦ أو ١٠٢٤) .

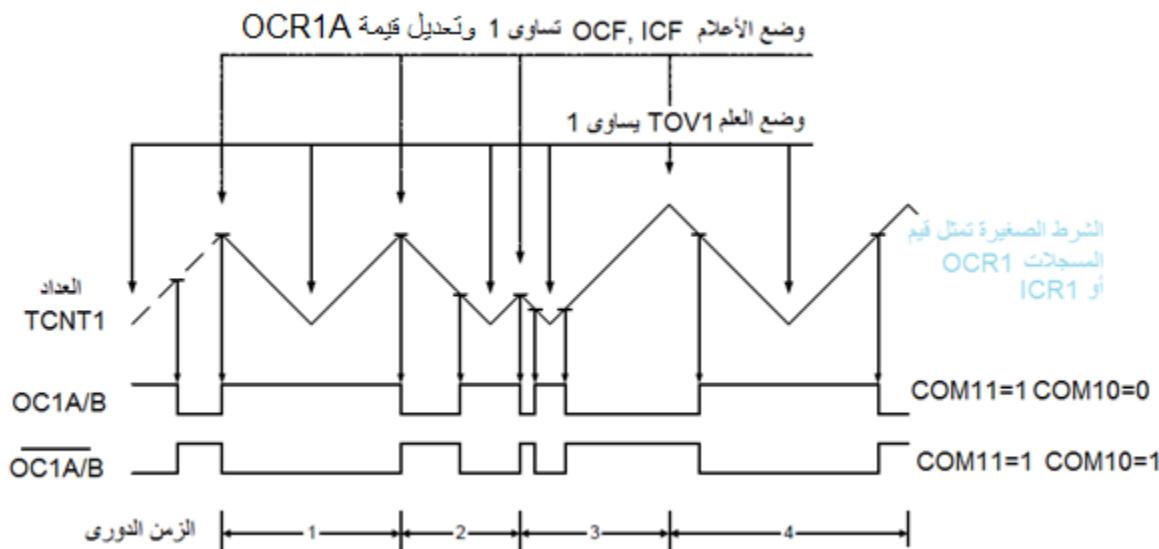
يمكن تنفيذ نفس برنامج تعديل عرض الموجة السريع PWM الموجود في الفصل ٨ مع مراعاة الفروق في قيم المسجلات الحاكمة لشكل الموجة في هذه الحالة.

#### ٤- طريقة تعديل عرض النسبة phase correct للمعدلة الطور PWM

هذه الطريقة من التشغيل يمكن الحصول عليها بوضع بتات الشكل الموجي WGM10 و WGM11 و WGM12 و WGM13 تساوى ١ أو ٢ أو ٣ أو ١٠ أو ١١ كما في الجدول ٢-٩ ، حيث تختلف هذه الحالات باختلاف القيمة العظمى التي يصل إليها العداد ومصدرها، وعدد بتات العداد المستخدمة. في هذه الطريقة يبدأ العداد في العد التصاعدى من الصفر إلى القيمة العظمى، ثم العد التنازلى من القيمة العظمى إلى الصفر مرة أخرى، وهكذا. لذلك فإن هذه الطريقة تعرف بالليل المزدوج.

في حالة التشغيل غير العاكسة يتم تصفير الطرف OC1A/B عند حدوث التساوى بين العداد TCNT1 ومسجل خرج المقارنة OCR1A/B أثناء صعود العداد، ويتم وضع هذا الطرف بوحد عند التساوى بين العداد ومسجل خرج المقارنة أثناء نزول العداد. في حالة التشغيل العاكسة يحدث عكس الخطوات السابقة.

في حالة التشغيل ١ كما في جدول ٢-٩ يكون عدد بتات العداد يساوى ٨ بت وبالتالي فإن القيمة العظمى للعداد ستكون  $TOP_d = 256 = 0x00FF$  ، وفي الحالة ٢ يكون عدد بتات العداد يساوى ٩ بت وبالتالي ستكون القيمة العظمى  $TOP_d = 512 = 0x01FF$  . في الحالة ٣ يكون عدد بتات العداد يساوى ١٠ بت، وبالتالي تكون القيمة العظمى  $TOP_d = 1024 = 0x03FF$  . في الحالة ١٠ تكون القيمة العظمى مخزنة في مسجل مسک الدخل ICR1 وفي الحالة ١١ تكون القيمة العظمى مخزنة في مسجل خرج المقارنة OCR1A/B . في الحالة ١٠ تكون القيمة العظمى مخزنة في مسجل مسک الدخل ICR1 وفي الحالة ١١ تكون القيمة العظمى مخزنة في مسجل خرج المقارنة OCR1A/B . شكل ١٨-٩ يبين مخطط للتزامن الحادث على الطرف OC1A/B سواء غير العاكس أو العاكس.



شكل ١٨-٩ التزامن الحادث مع طريقة تعديل عرض النبضة PWM المعدلة الطور (نسخة من دليل المتحكم (atmega328

تردد الموجة المعدلة العرض PWM ذات الطور المعدل يمكن الحصول عليها من المعادلة التالية:

$$f_{OC0APWM} = \frac{f_{clock}}{2 \cdot N \cdot TOP}$$

حيث  $f_{clock}$  هي نبضات تزامن المتحكم، و  $N$  هي نسبة القسمة المستخدمة للحصول على نبضات التزامن التي سيعمل عندها المؤقت (١ أو ٨ أو ٦٤ أو ٢٥٦ أو ١٠٢٤).

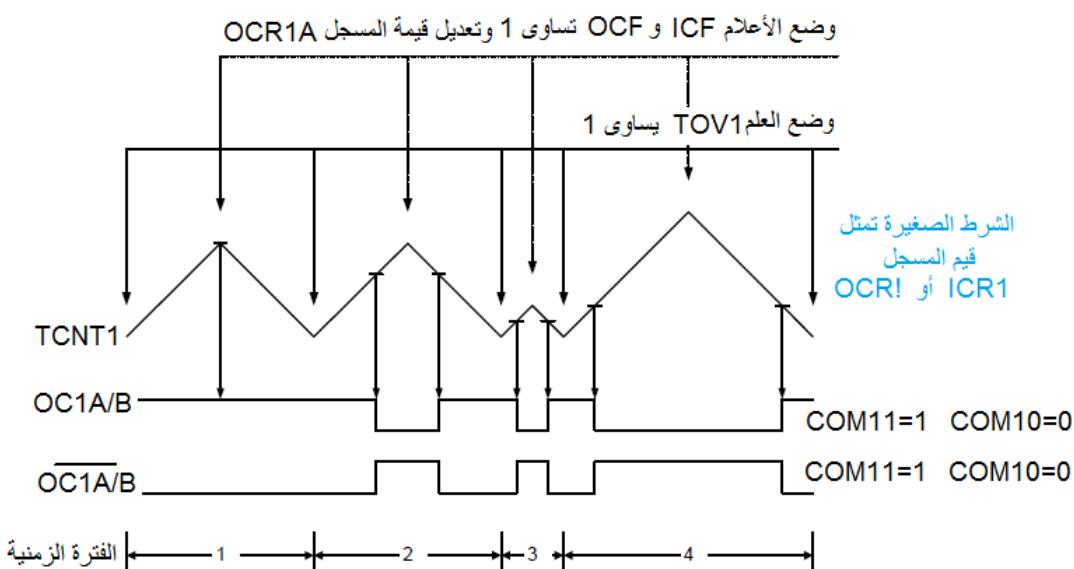
يمكن تنفيذ نفس برنامج تعديل عرض الموجة PWM المعدل الطور الموجود في الفصل ٨ مع مراعاة الفروق في قيم المسجلات الحاكمة لشكل الموجة في هذه الحالة.

## ٥-طريقة تعديل عرض النبضة PWM المعدلة الطور والتعدد

هذه الطريقة مثلاً تماماً مثل الطريقة السابقة المعدلة الطور فيما عدا لحظة تغيير المسجل **OCR1A** إذا كانت ستمثل القمة **TOP**. كما رأينا في شكل ٧-٩ فإن المسجل **OCR1A** له عازل بحيث عند تغيير قيمته، فإن القيمة الجديدة توضع في مسجل العزل أولاً إلى أن يصل العداد **TCNT1** إلى القيمة العظمى وتم المقارنة بما حيث عند ذلك تنتقل القيمة الجديدة من مسجل العزل إلى المسجل **OCR1A**. لذلك فإن التغيير الحقيقي لقيمة المسجل **OCR1A** لا يتم إلا عندما يصل العداد **TCNT1** إلى القمة. في حالة الطور المعدل، بعد تغيير قيمة المسجل **OCR1A** عند

قمة العداد فإنه يبدأ في النزول إلى الصفر وتم المقارنة أثناء النزول على القيمة القديمة قبل التغيير. عندما يبدأ العداد في الصعود فإنه سيظل يصعد إلى القيمة الجديدة TOP بعد التعديل. لذلك فإن هذه الدورة سيكون زمن نزولها مختلفاً عن زمن صعودها، وهذا هو الغيب في طريقة تعديل عرض النبضة المعدلة الطور.

في حالة تعديل عرض النبضة المعدلة الطور والتردد، فإنه عند تغيير قيمة المسجل فإن القيمة الجديدة لا يتم نقلها من العازل إلى المسجل OCR1A إلى عند وصول العداد للصفر، لذلك فإن الدورة هنا تبدأ وتنتهي عند القيمة الصغرى للعداد (صفر) على عكس الحالة السابقة التي كانت تبدأ فيها الدورة وتنتهي عند القيمة العظمى للعداد. هذا الوضع الجديد جعل زمن صعود العداد يساوى تماماً زمن نزوله أثناء الدورة، ولذلك تكون الدورة متماثلة الجانبين، وهذا هو سبب تسمية هذه الطريقة بأنها أصبحت معدلة التردد بجانب تعديل الطور. شكل ١٩-٩ يبين التزامن في هذه الطريقة.



شكل ١٩-٩ التزامن الحادث مع طريقة تعديل عرض النبضة PWM المعدلة الطور والتردد (نسخة من دليل المتحكم (atmega328

## ٦- تشغيل المؤقت ١ كعداد

باختيار مصدر نبضات المؤقت من الطرف T1 للمؤقت وهو الطرف رقم 11 في الشريحة، فإن المؤقت سيعمل كعداد يعد النبضات الداخلة على هذا الطرف. جدول ٥-٩ يبين المصادر المختلفة للنبضات، حيث باختيار البتات CS12=1 و CS11=1 و CS10=0 فإن العداد سيعمل في هذه الحالة مع الحافة النازلة للنبضات، وبوضع هذه

البتات  $CS10=1$  و  $CS11=1$  و  $CS12=1$ ، فإن العداد سيعمل مع الحافة الصاعدة للنبضات الداخلية على الطرف T1. طبعاً سيكون أقصى عدد يمكن أن يصل له العداد في هذه الحالة هو ٦٥٥٣٦ حيث أن المؤقت ١٦ بت. يمكن تنفيذ نفس برنامج تشغيل المؤقت ١ كعداد موجود في الفصل ٨ مع مراعاة الفروق في قيم المسجلات الحاكمة لشكل الموجة في هذه الحالة.

بذلك تكون قد انتهينا من تقديم كل ما يتعلق بتشغيل المؤقت ١ في الشريحة atmega328، وكما رأينا فإن تشغيل هذا المؤقت مشابه تماماً للمؤقت صفر، مع الفروق البسيطة التي ظهرت نتيجة كونه ١٦ بت، والتي يجب مراعاتها عند التشغيل.

## ملخص الفصل

يعتبر هذا الفصل امتداداً للفصل السابق مع فرق كون المؤقت ١ يتكون من ١٦ بت بدلاً من ٨ بت كما في المؤقت صفر. ولذلك سيكون التشابه كبيراً جداً في طريقة التقديم لكل منهما. من الممكن أن يقول البعض أنه كان يكفي الإشارة إلى الفروق الجديدة في المؤقت ١ في عنوان جانبي في الفصل السابق، ولكننا فضلنا أن نشرح المؤقت ١ بالتفصيل حتى يكون هذا الفصل مستقلاً عن سابقه. إننا فقط رأينا ألا نكرر كتابة البرامج من الفصل السابق لهذا الفصل وتركنا ذلك كتدريب للقارئ على إعادة هذه البرامج بالتعديل الذي يتواافق مع المؤقت ١.

# الفصل ١٠



## المؤقتات ... المؤقت ٢

### Timers ... Timer 2

**العناوين المضيئة في هذا الفصل:**

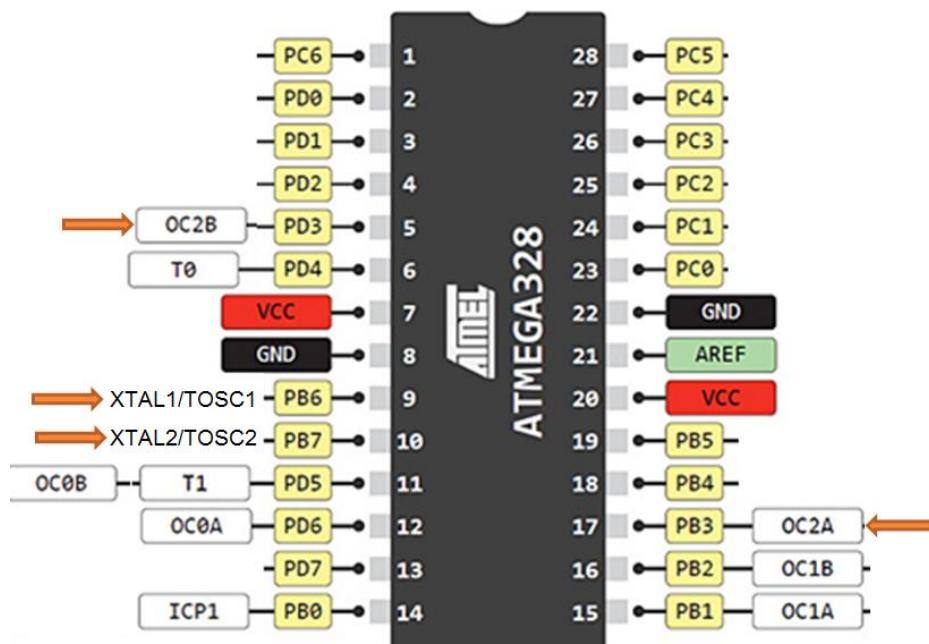
- ١- مصادر نبضات المؤقت ٢
- ٢- مسجل المؤقت ٢
- ٣- مسجل مقارنة الخرج A
- ٤- مسجل مقارنة الخرج B
- ٥- مقاطعة المؤقت ٢
- ٦- مسجلات التحكم في أداء المتحكم ٢
- ٧- حالات تشغيل المؤقت ٢
- ٨- التشغيل غير المتزامن للمؤقت ٢

## ١-١٠ مقدمة

**لقد** رأينا في الفصلين الثامن والتاسع تفاصيل تشغيل المؤقتين صفر و ١، ورأينا كيف أن المؤقت صفر كان يتكون من ٨ بت، بينما المؤقت ١ فكان يتكون من ١٦ بت. ورأينا أيضاً كيف أن المؤقت ١ يتكون من ١٦ بت بينما مسار البيانات للمتحكم يتكون في الأصل من ٨ بت فقط، لذلك كان لابد من وسيلة معينة للتعامل مع هذا المؤقت من خلال مسار البيانات، وكان هذا هو الفرق الجوهرى بين المؤقت صفر والمؤقت ١، بالإضافة طبعاً إلى الزيادة في طرق التشغيل وإن كانت الفروق بينها بسيطة إلى حد ما.

المؤقت ٢ يتكون من ٨ بت (مثل المؤقت صفر)، ولذلك فهو يشبه في تشغيله وطريقة التعامل معه المؤقت صفر بدرجة كبيرة جداً وسنرى ذلك من خلال هذا الفصل. سنشرح في هذا الفصل كما فعلنا مع الفصل ٩ حيث ركنا الفروق بين المؤقت صفر والمؤقت ١، ووجهنا القارئ إلى تنفيذ التمارين الموجودة في الفصل ٨ حتى لا يكون هناك تكرار لها، ونخن هنا سنتبع نفس الطريقة.

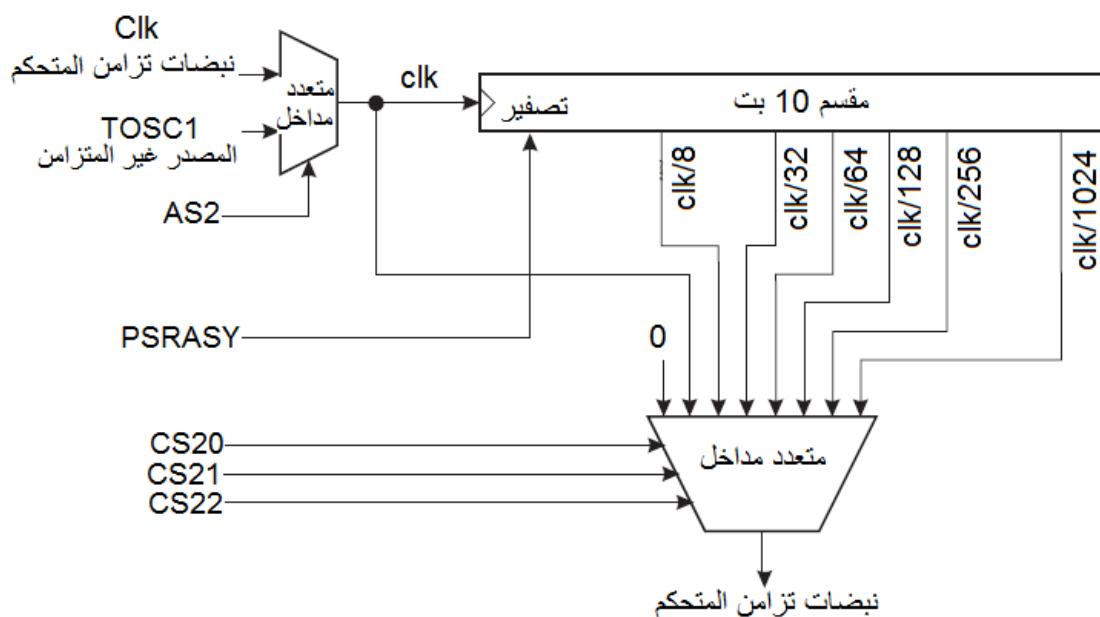
شكل ١-١٠ يبين أطراف المتحكم atmega328 المستخدمة مع المؤقت ٢ والتي سندرسها بالتفصيل في هذا الفصل.



شكل ١-١٠ أطراف المتحكم atmega328 المستخدمة مع المؤقت ٢

## ٢-١٠ مصادر نبضات التزامن للمؤقت ٢

يمكن للمؤقت ٢ أن يحصل على نبضات التزامن الخاصة به بترددات مختلفة إما خارجياً من على أطراف شريحة المتحكم، وهذا الطرفان ٩ و ١٠ في شريحة المتحكم وهو ما يمثلان المصدر غير المترافق للنبضات Asynchronous source، ويسميان TOSC1 و TOSC2 كما في شكل ١-١٠، وستتكلّم عن هذا المصدر بعد قليل. بالإضافة إلى ذلك يوجد مراحل قسمة عديدة لنبضات تزامن المتحكم الأساسية حيث يتم الاستفادة من خروج هذه المراحل كنبضات تزامن للمؤقت. يتم استخدام متعدد مداخل multiplexer لاختيار أحد هذه المصادر من خلال بิตات في أحد المسجلات كما سنرى. شكل ٢-١٠ يبيّن هذه المصادر المختلفة المستخدمة.



شكل ٢-١٠ المصادر المختلفة لنبضات تزامن المؤقت ٢

كما نلاحظ في شكل ٢-١٠ فإنه بوضع البٽ AS2 في المسجل ASSR تساوى ١، فإن المصدر غير المترافق TOSC1 سيدخل إلى مقسم النبضات كما في الشكل، وفي هذه الحالة يمكن استخدام المؤقت كعداد حقيقي للنبضات القادمة على هذا الطرف. وبوضع البٽ AS2 تساوى صفر فإن نبضات تزامن المتحكم الأصلية والمترافقه هي التي ستدخل إلى مقسم النبضات. بالنسبة للنبضات المترافقه فهي مخرج مقسم التردد حيث يتم قسمة نبضات ساعة المتحكم على ٣٢ أو ٦٤ أو ١٢٨ أو ٢٥٦ أو ١٠٢٤، أو تدخل نبضات التزامن كما هي بدون قسمة، أو يتم إدخال ٠ وفي هذه الحالة يتوقف العداد.

## ٣-١٠ مسجل المؤقت ٢

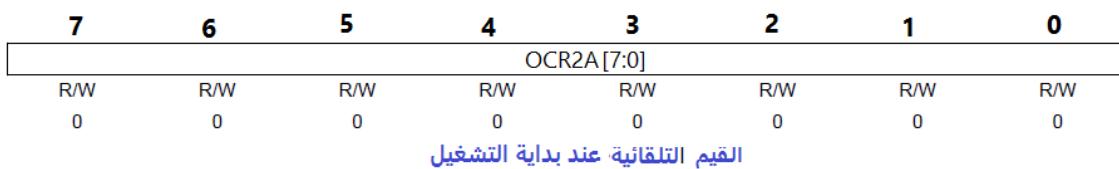
مسجل المؤقت ٢ Timer/counter2، TCNT2 يتكون من ثمانية بิตات ويحتوى قيمة العداد بعد كل نبضة تزامن، ولذلك فقيمة تراوح بين الصفر و ٢٥٥ لهذا المؤقت. هذا المسجل يمكن قراءته أو الكتابة فيه في أى وقت. بعد كل نبضة تزامن تزيد قيمة هذا العداد بمقدار واحد وتم مقارنته بمسجلات المقارنة compare register التي سيتم شرحها في الجزء التالى، وعلى ضوء هذه المقارنة يمكن اتخاذ كثيرة من الأفعال كما سنرى فيما بعد أيضا. شكل ٣-١٠ يبين رسمياً تخطيطياً لهذا المسجل حيث نلاحظ أنه يتكون من ٨ بิตات يمكن قراءتها وكتابتها في أى وقت.



شكل ٣-١٠ رسم تخطيطي لمسجل المؤقت ٢

## ٤-١٠ مسجل مقارنة الخرج A

يتكون مسجل مقارنة الخرج A للمؤقت ٢ output compare register A، OCR2A من ثمان بิตات أيضاً، ويتم فيه تسجيل القيمة التي سيتم مقارنتها مع محتويات مسجل المؤقت ٢ TCNT2 بعد كل نبضة تزامن، وعلى ضوء هذه المقارنة، وعند تساوى القيمتين، يتم إخراج إشارة معينة على طرف خرج المقارنة OC2A رقم ١٧ للمتحكم كما في شكل ٤-١٠، وسيأتي تفصيل لهذا الخرج فيما بعد. كما يمكن على ضوء هذا التساوى تنشيط علم مقاطعة والقفز إلى برنامج خدمة مقاطعة ISR. شكل ٤-١٠ يبين رسمياً تخطيطياً لهذا المسجل.



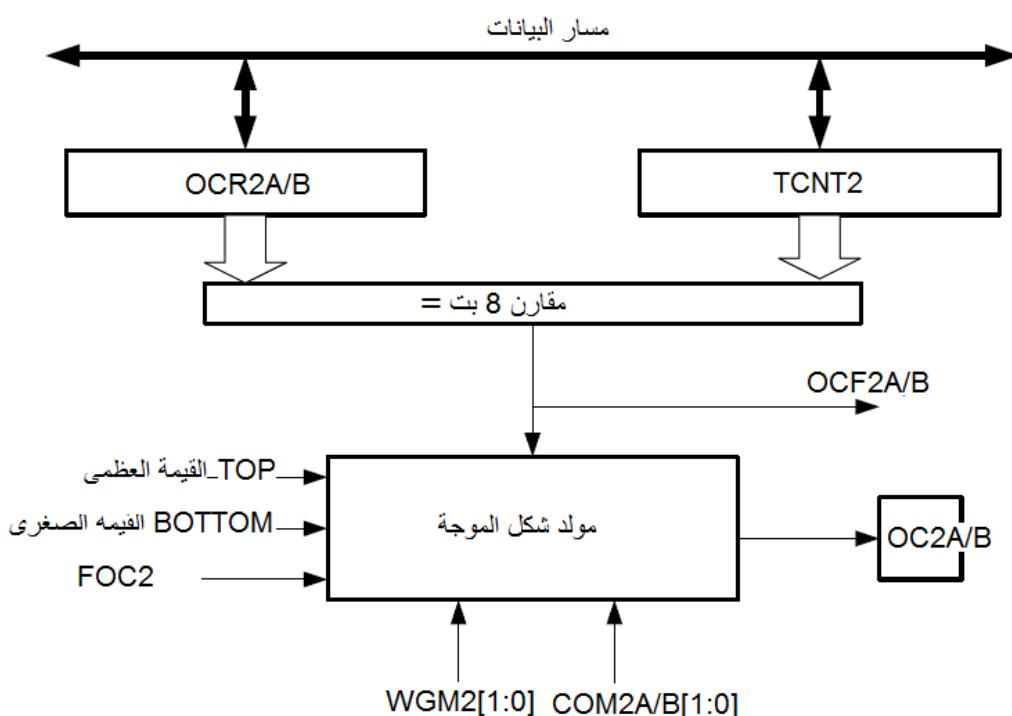
شكل ٤-١٠ مسجل مقارنة الخرج A للمؤقت ٢

## ٥-١٠ مسجل مقارنة الخرج B

يتكون مسجل مقارنة الخرج B output compare register B، OCR2B للمؤقت ٢ من ثمان برات أيضاً، ويتم فيه تسجيل القيمة التي سيتم مقارنتها مع محتويات مسجل المؤقت ٢ TCNT2 بعد كل نبضة تزامن، وعلى ضوء هذه المقارنة، وعند تساوى القيمتين، يتم إخراج إشارة معينة على طرف خرج المقارنة OCR2B رقم 5 للمتحكم كما في شكل ١-١٠، وسيأتي تفصيل لهذا الخرج فيما بعد. كما يمكن على ضوء هذا التساوى تنشيط علم مقاطعة والقفز إلى برنامج خدمة مقاطعة ISR. شكل ٥-١٠ يبين رسمياً تخطيطياً لهذا المسجل.



شكل ٥-١٠ مسجل مقارنة الخرج B للمؤقت ٢



شكل ٦-١٠ رسم تخطيطي لعملية المقارنة في المؤقت ٢ (نسخة من دليل المتحكم atmega328

شكل ٦-١٠ يبين رسمياً تخطيطياً لعملية مقارنة الخرج في المؤقت ٢. نلاحظ من هذا الشكل أن هناك مقارن ٨ بت يقوم بمقارنة بتات مسجل المؤقت ٢ TCNT2 مع بتات مسجل المقارنة OCR2A أو OCR2B حيث يقصد بها A أو B) بحيث أنه عند تساوى القيمتين فإنه إما أن يتم تنشيط علم مقاطعة خاص بالمقارنة A وهو OCF2A أو المقارنة B وهو OCF2B تمهداً للانتقال إلى برنامج خدمة مقاطعة إذا كانت هذه المقاطعة فعالة وكان علم المقاطعة العام في مسجل الأعلام فعالاً أيضاً، أو أنه يتم إخراج موجة معدلة العرض PWM على أي من الطرفين OC2B أو OC2A من أطراف شريحة المتحكم كما أشرنا مسبقاً.

## ٦-١٠ مقاطعة المؤقت ٢

المؤقت ٢ له ثلات مصادر للمقاطعة كما هو مبين في الجدول ١-١٠ الذي يوضح مصدر كل مقاطعة، والعنوان الذي يتم القفز إليه عند حدوث المقاطعة، ورقم متوجه المقاطعة. المقاطعة الأولى تقع عند حدوث تساوى بين محتويات مسجل المؤقت ٢ TCNT2 ومسجل مقارنة الخرج OCR2A. عند وقوع هذا التساوى بشرط أن يكون قناع المقاطعة الخاص بهذه المقاطعة نشطاً، وأن يكون علم المقاطعة العام في مسجل الأعلام نشطاً، فإن المتحكم سيقفز إلى العنوان 0x007 كما في الجدول.

جدول ١-١٠ إسم ومصدر وعنوان ورقم متوجه المقاطعات في المؤقت ٢ (نسخة من دليل المتحكم)

رقم المتوجه	عنوان المتوجه	مصدر المقاطعة	اسم المقاطعة
8	0x007	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x008	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x009	TIMER2 OVF	Timer/Counter2 Overflow
11	0x00A	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x00B	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x00C	TIMER1 COMPB	Timer/Counter1 Compare Match B
14	0x00D	TIMER1 OVF	Timer/Counter1 Overflow
15	0x00E	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x00F	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x010	TIMER0 OVF	Timer/Counter0 Overflow

المقاطعة الثانية تقع عند حدوث تساوى بين محتويات مسجل المؤقت ٢ TCNT2 ومسجل مقارنة الخرج OCR2B. عند وقوع هذا التساوى بشرط أن يكون قناع المقاطعة الخاص بهذه المقاطعة نشطاً، وأن يكون علم المقاطعة العام في

مسجل الأعلام نشطا، فإن المتحكم يقفز إلى العنوان 0x008 كما في الجدول. المقاطعة الثالثة تقع عند حدوث فيضان في مسجل المؤقت 2، يعني أن تصل محتوياته إلى 255، حيث عندها يقفز المتحكم إلى العنوان 0x009 بشرط أن يكون كل من قناع هذه المقاطعة وقناع المقاطعة العام في مسجل الأعلام نشطين.

### مسجل أقنعة مقاطعات المؤقت ٢ Timer Interrupt Mask Register, TIMSK2

يبين شكل ٧-١٠ برات هذا المسجل، حيث نلاحظ استخدام أول ثلاثة برات فقط من هذا المسجل وهي كالتالي:

7	6	5	4	3	2	1	0
0	0	-	-	-	OCIEB	OCIEA	TOI

القيم التلقائية عند بداية التشغيل

شكل ٧-١٠ مسجل أقنعة مقاطعات المؤقت ٢

**البت ٠: قناع مقاطعة الفيضان Timer Over flow Interrupt Enable, TOIE**، حيث بوضع واحد في هذه البت يتم تنشيط مقاطعة الفيضان ويقفز المتحكم إلى العنوان 0x009 إذا كان علم المقاطعة العام I يساوى واحد أيضا.

**البت ١: قناع مقاطعة مقارنة الخرج Output Compare Interrupt Enable A, OCIEA**، حيث بوضع واحد في هذه البت يتم تنشيط مقاطعة متساوية العداد 2 TCNT2 ومسجل مقارنة الخرج A ويقفز المتحكم إلى العنوان 0x008 إذا كان علم المقاطعة العام I يساوى واحد.

**البت ٢: قناع مقاطعة مقارنة الخرج OCIEB**، حيث بوضع واحد في هذه البت يتم تنشيط مقاطعة متساوية العداد 2 TCNT2 ومسجل مقارنة الخرج B ويقفز المتحكم إلى العنوان 0x007 إذا كان علم المقاطعة العام I يساوى واحد.  
باقي برات هذا المسجل غير مستخدمة.

في كل المقاطعات الثلاث السابقة كيف يعرف المتحكم بحدوث تساوى بين مسجلات المقارنة A أو B أو حدوث فيضان في محتويات العداد. يتم ذلك عن طريق علم مقاطعة خاص بكل منها في المسجل التالى.

### مسجل أعلام مقاطعة المؤقت ٢ Timer/Counter0 Interrupt Flag Register, TIFR2

يبين شكل ٨-١٠ ببات هذا المسجل، حيث نلاحظ استخدام أول ثلاثة ببات فقط من هذا المسجل وهى كالتالى:

7	6	5	4	3	2	1	0
0	0	-	-	-	OCFB	OCFA	TOV

القيم التلقائية عند بداية التشغيل

شكل ٨-١٠ مسجل أعلام مقاطعات المؤقت ٢

**البت ٠: علم حدوث الفيضان Timer Over flow Flag, TOV**، هذه البت (العلم) تصبح واحد عند حدوث فيضان في محتويات عداد المؤقت ٢ TCNT2، ويتم تصفيرها عن طريق المتحكم بمجرد القفز إلى برنامج خدمة المقاطعة ISR الخاص بها.

**البت ١: علم مقارنة الخرج Output Compare A Interrupt Flag, OCF2A**، هذه البت (العلم) تصبح واحد عند تساوى العداد ٢ TCNT2 ومسجل مقارنة الخرج A، ويتم تصفيرها عن طريق المتحكم بمجرد القفز إلى برنامج خدمة المقاطعة ISR الخاص بها.

**البت ٢: علم مقارنة الخرج Output Compare B Interrupt Flag, OCF2B**، هذه البت (العلم) تصبح واحد عند تساوى العداد ٢ TCNT2 ومسجل مقارنة الخرج B، ويتم تصفيرها عن طريق المتحكم بمجرد القفز إلى برنامج خدمة المقاطعة ISR الخاص بها.  
باقي ببات هذا المسجل غير مستخدمة.

إذن باختصار يمكننا أن نلخص آلية المقاطعات الثلاث السابقة بأنه بمجرد حدوث تساوى بين مسجل عداد المؤقت ٢ TCNT2 وأحد مسجلات المقارنة A أو B، أو حدوث فيضان في محتويات المسجل المسجل المقابل

لكل منهم TOV أو OCF2A أو OCF2B يصبح واحد، وبالتالي إذا كان قناع المقاطعة المقابل لكل واحدة من هذه المقاطعات OCIEB أو OCIEA أو TOIE يساوى واحد، وإذا كان علم المقاطعة العام I في مسجل الحالة يساوى واحد أيضاً، فإن المقاطعة ستحدث، ويقفز المتحكم إلى برنامج خدمة المقاطعة ISR المقابل لأى منهم، وبحرجد انتقال المتحكم إلى برنامج خدمة المقاطعة ISR فإن العلم المقابل يتم تصفيره مرة ثانية استعداداً لحدوث مقاطعة أخرى.

## ٧-٨ مسجلات التحكم في أداء المؤقت ٢

يتم التحكم في أداء المؤقت ٢ عن طريق مسجلين A و B لهذا الغرض، وسنعرض بعده كل من هذين المسجلين بالتفصيل في هذا الجزء. شكل ٩-١٠ وشكل ١٠-١٠ يبيّنان ربما تخطيطياً لهذين المسجلين.

7	6	5	4	3	2	1	0
COM2A1	COM2A0	COM2B1	COM2B0			WGM21	WGM20
0	0						

القيم التلقائية عند بداية التشغيل

شكل ٩-١٠ المسجل TCCR2A للتحكم في المؤقت ٢

7	6	5	4	3	2	1	0
FOC2A	FOC2B			WGM22		CS2[2:0]	
0	0						

القيم التلقائية عند بداية التشغيل

شكل ١٠-١٠ المسجل TCCR2B للتحكم في المؤقت ٢

## مسجل التحكم A في المؤقت/العداد ٤ TCCR2A

البتات ٠ و ١ في مسجل التحكم TCCR2A والبت ٣ في مسجل التحكم TCCR2B، WGM20 و WGM21 و WGM22 : هذه البتات الثلاثة خاصة بالتحكم في تتبع عملية العد في العداد، وفي مصدر القيمة العظمى التي يمكن أن يصل إليها العداد، وفي نوع الموجة المولدة على طرف خرج المقارنة التي يمكن استخدامها، حيث يمكن استخدام واحد من ثمانية طرق أو ثمانية حالات للتشغيل أو للحصول على موجة بشكل معين مثل الموجة المربعة المعدلة العرض PWM كما سنرى بالتفصيل فيما بعد. جدول ٢-١٠ يبيّن هذه الحالات الثمانية للتشغيل في مقابل قيمة كل بت من هذه البتات.

**الآلات ٦ و ٧ في المسجل TCCR2A:** هذه الآلات COM2A0 و COM2A1 تتحكم في سلوك الخرج على طرف تساوى المقارنة OC2A، من حيث هل هذا الخط سيغير حالته مع كل تساوى بين مسجل المقارنة OCRA و المسجل TCNT2 أم أنه سيصبح صفرًا أم سيكون واحدًا. كل هذه الحالات يبينها الجدول ٣-١٠. لاحظ أن الطرف OC2A وهو الطرف ١٧ في شريحة المتحكم لابد أن يتم تشغيله كطرف خرج من خلال مسجل الاتجاه PDDR. الحالات الموجودة في الجدول ٣-١٠ للطرف OC2A تكون حالات التشغيل التي لا يكون فيها تعديل موجي على عرض الموجة الخارجية PWM حيث في هذه الحالة ستعمل هذه الآلات بطريقة أخرى سنشرحها بعد قليل مع شرح حالات التشغيل الثمانية.

جدول ٢-١٠ وصف آلات تشغيل المؤقت ٢، لاحظ أن MAX=0xFF و Bottom=0x00

(نسخة من دليل المتحكم)

الحالة Mode	WGM02	WGM01	WGM00	حالات التشغيل Modes of operation	القيمة العظمى TOP	تجدد قيمة مسجل المقارنة عند،	لحظة وضع علم الفيضان بواحد
0	0	0	0	العادى	0xFF	فورى	MAX
1	0	0	1	موجة معدلة العرض، تصحيح الطور PWM, phase correct	0xFF	TOP	Bottom
2	0	1	0	CTC	OCRA	فورى	MAX
3	0	1	1	Fast PWM	0xFF	Bottom	MAX
4	1	0	0	محجوز، غير مستخدم	-----	-----	-----
5	1	0	1	موجة معدلة العرض، تصحيح الطور PWM, phase correct	OCRA	TOP	Bottom
6	1	1	0	محجوز، غير مستخدم	-----	-----	-----
7	1	1	1	Fast PWM	OCRA	Bottom	TOP

**الآلات ٤ و ٥ في المسجل TCCR2A:** هذه الآلات COM2B0 و COM2B1 تتحكم في سلوك الخرج على طرف تساوى المقارنة OC2B، من حيث هل هذا الخط سيغير حالته مع كل تساوى بين مسجل المقارنة OCRB و المسجل TCNT2 أم أنه سيصبح صفرًا أم سيكون واحدًا. كل هذه الحالات يبينها الجدول ٤-١٠. لاحظ أن

الطرف OC2B وهو الطرف 5 في شريحة المتحكم لابد أن يتم تشغيله كطرف خرج من خلال مسجل الاتجاه PDDR. الحالات الموجودة في الجدول ٤-١٠ للطرف OC2B تكون حالات التشغيل التي لا يكون فيها تعديل موجي على عرض الموجة الخارجة PWM حيث في هذه الحالة ستعمل هذه البتات بطريقة أخرى سنشرحها بعد قليل مع شرح حالات التشغيل الثمانية.

جدول ٣ حالات خرج المقارنة مع مسجل المقارنة OCR2A وفي غير حالة التشغيل PWM  
(نسخة من دليل المتحكم)

وصف الحالة	COM2A1	COM2A0
الطرف OC2A يعمل كطرف بوابة عادي، ليس له علاقة بالمقارنة Normal port operation	0	0
عند حدوث التساوى يغير الخط OC2A من حالته، فإذا كان صفر يصبح واحد، وإذا واحد يصبح صفر. Toggle on compare match	0	1
عند حدوث التساوى يصبح الطرف OC2A صفر، Clear on compare match	1	0
عند حدوث التساوى يصبح الطرف OC2A واحد، Set on compare match	1	1

جدول ٤-٤ حالات خرج المقارنة مع مسجل المقارنة OCR2B وفي غير حالة التشغيل PWM  
(نسخة من دليل المتحكم)

وصف الحالة	COM2B1	COM2B0
الطرف OC2B يعمل كطرف بوابة عادي، ليس له علاقة بالمقارنة Normal port operation	0	0
عند حدوث التساوى يغير الخط OC2B من حالته، فإذا كان صفر يصبح واحد، وإذا واحد يصبح صفر. Toggle on compare match	0	1
عند حدوث التساوى يصبح الطرف OC2B صفر، Clear on compare match	1	0
عند حدوث التساوى يصبح الطرف OC2B واحد، Set on compare match	1	1

**البتات ٤ و ٥ في المسجل TCCR2A:** هذه البتات هى COM2B1 و COM2B0، وهذه البتات تتحكم في سلوك الإشارة على خط خرج المقارنة OC2B. إذا كان أى واحد من كل من البت COM2B0 والبت COM2B1 أو كليهما تساوى واحد، فإن وظيفة الطرف OC2B ستكون هي السائدة ولن يعمل هذا الطرف

طرف خرج أو دخل من أطراف البوابة الملحق بها، وعلى الرغم من ذلك فإن هذا الطرف لابد أن يتم تخصيصه كطرف خرج من خلال أطراف مسجل الاتجاه DDR الخاص بهذه البوابة. جدول ٤-١٠ يبين علاقة طرف الخرج OC2B مع البات COM2B[1:0]، مع حالات التشغيل التي لا تكون معدلة العرض PWM.

**البات ٢ و ٣ في المسجل TCCR0A غير مستخدمة.**

### مسجل التحكم B في المؤقت/العداد ٢ TCCR2B

**البات ٠ و ١ و ٢ في المسجل TCCR2B:** هذه البات CS20 و CS21 و CS22 يتم عن طريقها اختيار مصدر نبضات تزامن المؤقت ٢ تبعاً للجدول ٥-١٠.

جدول ٥-١٠ اختيار مصدر نبضات التزامن للمؤقت ٢ (أنظر شكل ٢) (نسخة من دليل المتحكم)

CS22	CS21	CS20	الوصف
0	0	0	لا يوجد مصدر، ويتوقف المؤقت No clock source
0	0	1	نفس نبضات المتحكم، معامل قسمة يساوى ١ ، No prescaling
0	1	0	نبضات المتحكم على ٨
0	1	1	نبضات المتحكم على ٣٢
1	0	0	نبضات المتحكم على ٦٤
1	0	1	نبضات المتحكم على ١٢٨
1	1	0	نبضات المتحكم على ٢٥٦
1	1	1	نبضات المتحكم على ١٠٢٤

**البت ٧ في المسجل TCCR2B:** هذه البت FOC2A، عندما تكون واحد تسبب في إحداث مقارنة فورية بين مسجل المقارنة OCRA و المسجل TCNT2 والتأثير على طرف خرج المقارنة OC2A تبعاً للأطراف COM2A1 و COM2A0 الموضحتين في جدول ٣-١٠. هذه البت لا تعمل إلا مع حالات التشغيل التي لا يكون فيها تعديل لعرض الموجة PWM، أي أن يجب أن يكون معملاً عن طريق بات شكل موجة التعديل WGM، لذلك يجب أن تكون هذه البت صفر في حالة العمل في هذه الحالات من التشغيل.

**البت ٦ في المسجل TCCR2B:** هذه البت FOC2B، عندما تكون واحد تسبب في إحداث مقارنة فورية بين مسجل المقارنة OCRB و المسجل TCNT2 والتأثير على طرف خرج المقارنة OC2B تبعاً للأطراف

فيها تعديل لعرض الموجة PWM، أى أن يكون معطلا عن طريق بتات شكل موجة التعديل WGM، لذلك يجب أن تكون هذه البت صفر في حالة العمل في هذه الحالات من التشغيل.

**البتات ٤ و ٥ في المسجل TCCR2B:** غير مستخدمة.

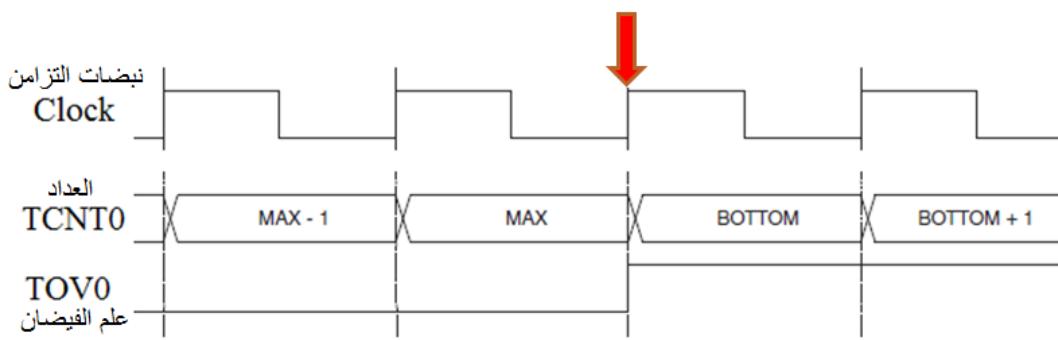
## ٨-١٠ حالات تشغيل المؤقت/العداد

تتحدد حالات التشغيل، أو أداء المؤقت على أطراف المقارنة OC2A أو OC2B على ضوء بتات اختيار الشكل الموجي على هذه الأطراف WGM في المسجلين TCCR2A و TCCR2B، والبتات COM2A0 و COM2A1 في المسجل TCCR2A. هناك ثمانية من هذه الحالات التي سنقدمها في هذا الجزء والتي تم عرضها في الجدول ٢-١٠.

### ١- حالة التشغيل العادية Normal mode

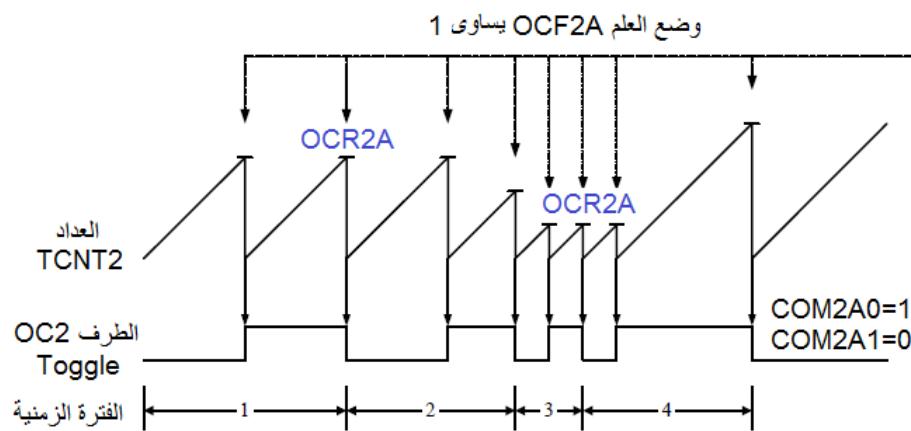
هذه الحالة من حالات التشغيل هي أبسط حالة في الحالات الثمانية. يتم الدخول في هذه الحالة بوضع بتات شكل الموجة WGM20 و WGM21 و WGM22 كلها بأصفار كما في الجدول ٢-١٠. في هذه الحالة يقوم العداد TCNT2 بالعد دائما في الاتجاه التصاعدي بزيادة مقدارها واحد، ولا يتم تصفير العداد نهائيا إلا عند وصوله للقيمة العظمى TOP=0xFF، حيث عندها يبدأ العداد في العد مرة ثانية من القيمة الصغرى Bottom=0x00. بمجرد أن تصبح قيمة العداد TCNT2 تساوى صفر ومع نفس نبضة التزامن فإن علم الفيضان TOV2 يصبح واحد (أى أن هذه البت تسلك مسلك البت التاسعة للعداد)، تمهيدا للقفز إلى برنامج خدمة المقاطعة ISR إذا كان قناع هذه المقاطعة وعلم المقاطعة العام نشطين كما أشرنا مسبقا. هذا العلم يتم تصفيره بمجرد القفز إلى برنامج خدمة المقاطعة ISR تمهيدا لدورة العد التالية. شكل ١١-١٠ يبين التزامن بين نبضات التزامن وقيمة العداد TCNT2 وعلم الفيضان، حيث نلاحظ أنه بمجرد وصول العداد للقيمة العظمى (السهم الأحمر) يرتفع علم الفيضان من الصفر إلى الواحد كما أشرنا.

فصل ٨ يحتوى تنفيذ برنامج على هذه الحالة من التشغيل مع المحاكاة على برنامج البروتوس، يرجى الرجوع له.



شكل ١١-١٠ التزامن بين نبضات التزامن والعداد TCNT2 وعلم الفيضان في حالة التشغيل العادي

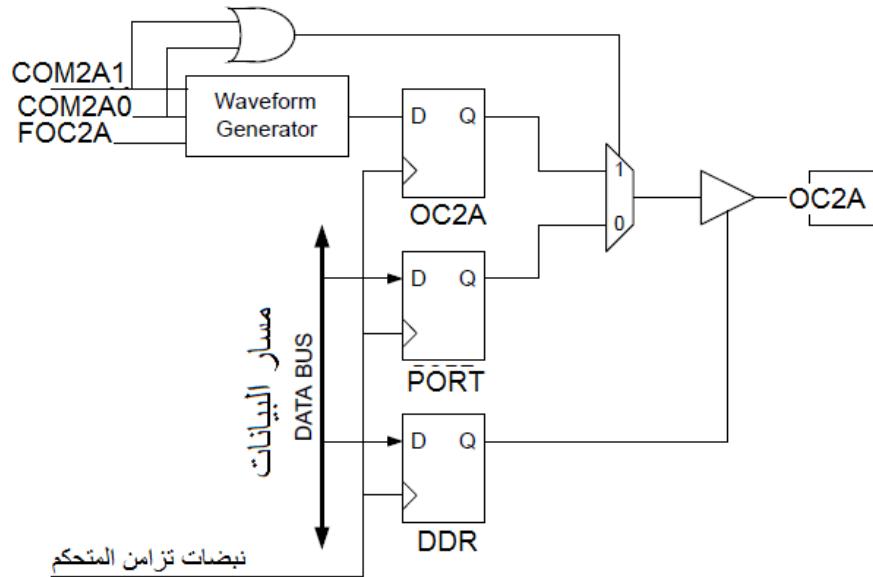
## ٢- تصفيير المؤقت ٢ عند تساوى المقارنة CTC



شكل ١٢-١٠ التزامن على الطرف OC2A مع قيمة مسجل المقارنة OCR2A (نسخة من دليل المتحكم)

في هذه الحالة، تصفيير المؤقت عند تساوى المقارنة، CTC، تكون الشفرة على برات توليد الموجة WGM20 و WGM22 و WGM21 تساوى الرقم ٢ الثنائي، ٠١٠، كما في الجدول ٢-١٠. يتم استخدام مسجل المقارنة OCR2A لتوضع فيه القيمة التي سيتم مقارنتها مع محتويات العداد TCNT2. في هذه الطريقة عندما يحدث التساوى بين الاثنين يتم تصفيير العداد ليبدأ دورة عد جديدة. معنى ذلك أن قيمة مسجل المقارنة OCR2A تكون هي القيمة العظمى للعداد، وعند حدوث هذا التساوى فإن علم المقاطعة OCF2A يصبح واحد، وبالتالي فإن المتحكم يمكن أن يقفر إلى برنامج خدمة المقاطعة ISR الخاص بهذا الحدث إذا كان كل من القناع الخاص بهذه المقاطعة OCF2A وعلم المقاطعة العام I نشطين، وبالطبع فإن علم المقاطعة OCF2A سيعود إلى الصفرة مرة ثانية بمجرد

الدخول في برنامج خدمة المقاطعة. شكل ١٢-١٠ يبين مخطط التزامن في هذه الحالة حيث نلاحظ أنه بمجرد وصول العداد TCNT2 إلى القيمة الموجودة في المسجل OCR2A فإن العداد يتم تصفيه. يمكن استخدام هذه الحالة في الحصول على شكل موجي متغير على الطرف OC2A عن طريق تعديل القيمة الموجودة في مسجل المقارنة في برنامج خدمة المقاطعة مع وضع البิตين COM2A1 و COM2A0 بحيث يغير هذا الطرف من حالته (toggle) عند كل تساوى، كما في جدول ٣-١٠.



شكل ١٣-١٠ علاقة طرف المقارنة OC2A مع طرف البوابة المقابل له (نسخة من دليل المتحكم)

لاحظ أن الإشارة على الطرف OC2A لن تكون مرئية إلا إذا كان هذا الطرف موضوع كطرف خرج عن طريق مسجل الاتجاه الخاص بالبوابة التابع لها هذا الطرف. يمكن تحديد تردد الموجة الناشئة على الطرف OC2A تبعاً للمعادلة التالية:

$$f_{OC2A} = \frac{f_{clock}}{2N(1+OCR2A)}$$

حيث N هي معامل القسمة (١ أو ٨ أو ٣٢ أو ٦٤ أو ١٢٨ أو ٢٥٦ أو ١٠٢٤) المستخدم للحصول على نبضات تزامن المؤقت، fclock هي نبضات تزامن المتحكم. لاحظ أن أكبر تردد يمكن الحصول عليه في هذه الحالة سيكون عندما  $OCR2A=0$ .

شكل ١٣-١٠ يبين العلاقة بين خرج المقارنة OC2A (ومثله تماماً الخرج OC2B) والبوابة التي هو أحد أطرافها. نلاحظ من هذا الشكل أن الطرف OC2A (في أقصى يسار الشكل) لكي يعمل هذه الوظيفة فإن خرج قلاب الاتجاه DDR يجب أن يكون واحد حتى ينشط العازل buffer الموصل لهذا الطرف بالخرج القادم من القلاب في أعلى الشكل. في هذه الحالة إذا كان أي واحد من البتات COM2A0 أو COM2A1 أو كليهما يساوى واحد فإن بوابة الأور ستعطى واحد وهذا الواحد سيمرر الدخل الأعلى (وهو الإشارة القادمة من مولد الموجات wave form generator) من متعدد المداخل إلى طرف الخرج OC2A. على النقيض من ذلك، إذا كان خرج بوابة الأور يساوى صفر (كل من COM2A0 و COM2A1 يساوى صفر في نفس الوقت)، فإن الدخل الأسفل من متعدد المدخل وهو خرج البوابة سيوصل على طرف الخرج، وفي هذه الحالة فإن الطرف OC2A لن يؤدي وظيفته، ولكنه سيصبح طرف خرج عادي من أطراف البوابة D. لذلك كان التأكيد فيما سبق على أنه لكي يؤدي طرف المقارنة OC2A أو OC2B وظيفته، فإن طرف البوابة المقابل يجب أن يتم تعينه على أنه طرف خرج بوضع واحد في قلاب الاتجاه المعاكس.

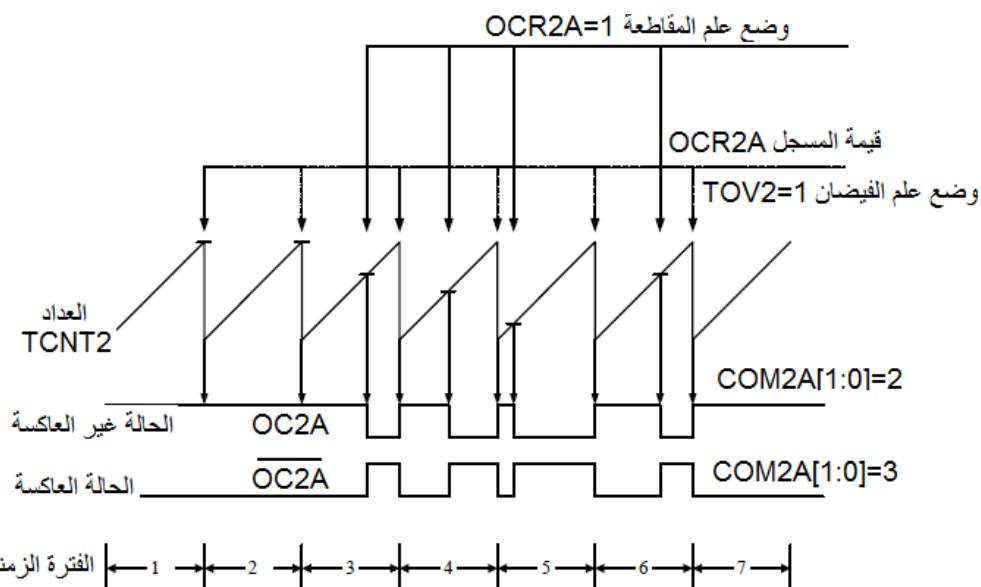
[يمكن تنفيذ برنامج تشغيل المؤقت ٢ في هذه الحالة \(تصغير المؤقت عند تساوى المقارنة CTC\) الموجود في الفصل ٨.](#)

### ٣-طريقة تعديل عرض النبضة PWM السريعة

في هذه الطريقة تكون بتات توليد الشكل الموجي تساوى ٣ (011) أو ٧ (111) كما في الجدول ٢-١٠. تتميز هذه الطريقة بوجود ميل واحد فقط وهو في حالة صعود العداد TCNT2 من صفر إلى القيمة العظمى TOP=0xFF وذلك في حالة أن بتات الشكل الموجي WGM20، WGM21، و WGM22 تساوى ٣ (011) ثم ينزل للصفر مرة ثانية ويبدأ في الصعود وهكذا. إذا كانت بتات الشكل الموجي تساوى ٧ (111)، فإن العداد يصعد من الصفر إلى القيمة العظمى TOP=OCR2A، أي إلى القيمة المخزنة في مسجل المقارنة ثم ينزل للصفر ويبدأ في الصعود مرة ثانية، وهكذا. هناك حالتان لظهور الموجة على طرف خرج المقارنة OC2A وهما: حالة عدم العكس وفيها يتم تصغير الطرف OC2A، عند لحظة تساوى قيمة العداد TCNT2 مع مسجل المقارنة OCR2A، ثم يتم إعادةه للواحد مرة ثانية عندما يصبح العداد صفرًا. في الحالة العاكسة يتم وضع طرف خرج المقارنة بواحد عند لحظة تساوى العداد TCNT2 ومسجل المقارنة OCR2A، ثم تصفيه عند وصول العداد للصفر. بسبب هذا الميل الوحيد وهو مع صعود العداد من الصفر للقيمة العظمى، فإن هذه الطريقة تكون أسرع (تقريباً ضعف) من الطريقة

التالية وهي طرق تعديل الطور phase correct المزدوجة الميل والتي سنشرحها بعد قليل. هذه السرعة تجعل هذه الطريقة مناسبة للكثير من التطبيقات من الطريقة الثانية.

شكل ١٤-١٠ يبين التزامن الحادث مع هذه الطريقة. في هذا الشكل تمثل الخطوط الأفقيّة القصيرة التي على إشارة العداد قيمة مسجل المقارنة التي ستتم عندها المقارنة. لاحظ في هذا الشكل أن العداد TCNT2 يعد دائماً من الصفر إلى القيمة العظمى 0xFF، وعندما يصل العداد لهذه القيمة يبدأ من الصفر مرة أخرى ويصبح علم الفيضان TOV2 بوحدة عند هذه اللحظة. في أثناء صعود العداد TCNT2 من الصفر إلى الواحد، وفي لحظة تساويه مع قيمة مسجل المقارنة OCR2A الممثلة بالخط الأفقي القصير، فإن طرف خرج المقارنة OC2A ينزل للصفر في حالة عدم الانعكاس (البتات COM2A0=1) أو يصعد من صفر لواحد في حالة الانعكاس (البتات COM2A1=0). وبعدها يعود طرف OC2A إلى صفر (البتات COM2A1=1)، ويظل طرف OC2A على هذه الحالة إلى أن يصل العداد للحالة العظمى 0xFF حيث يرجع طرف OC2A إلى أصله مرة ثانية، وهي الواحد في حالة عدم الانعكاس، أو الصفر في حالة الانعكاس. إذن يعني ذلك أن الزمن الدورى للموجة الناجمة على طرف OC2A سيكون ثابت، وأما نسبة الزمن OFF للزمن ON للزمن أثناء الزمن الدورى فسيتم التحكم فيها عن طريق القيمة الموجودة في مسجل المقارنة، ومن هنا كانت فكرة تعديل عرض الموجة PWM الموضحة في شكل ١٤-١٠. في الحالة العاكسة يقل الزمن ON بزيادة القيمة المخزنة في مسجل المقارنة OCR2A، وفي الحالة غير العاكسة، فإن الزمن ON يزيد بزيادة القيمة المخزنة في مسجل المقارنة OCR2A.



شكل ١٤-١٠ التزامن الحادث مع طريقة تعديل عرض النبضة PWM السريع في حالة المؤقت ٢

تذكر هنا أنه لكي تم رؤية هذه الموجات على طرف المقارنة OC2A، فإن هذا الطرف لابد أن يكون قد تم تعينه ليكون طرف خرج بوضع وحيد في مسجل اتجاه البوابة التابع لها هذا الطرف كما في شكل ١٣-١٠.

لاحظ أنه في نهاية كل زمن دورى فإن علم الفيضان TOV2 يتم وضعه بوحدة واحدة، وفي هذه الحالة إذا تم تنشيط كل من قناع المقاطعة TOIE2 وعلم المقاطعة العام I، فإنه يمكن القفز إلى برنامج خدمة مقاطعة ISR، ويمكن في هذا البرنامج تعديل قيمة مسجل المقارنة OCR2A على حسب الطلب.

تردد الموجة المعدلة العرض PWM السريع يمكن الحصول عليها من المعادلة التالية:

$$f_{OC2APWM} = \frac{f_{clock}}{N \cdot 256}$$

حيث  $f_{clock}$  هي نبضات ترامن المتحكم، و  $N$  هي نسبة القسمة المستخدمة للحصول على نبضات الترامن التي سيعمل عندها المؤقت (١ أو ٨ أو ٣٢ أو ٦٤ أو ١٢٨ أو ٢٥٦ أو ١٠٢٤).

وضع القيمة العظمى 0xFF في مسجل المقارنة ستجعل الطرف OC2A يساوى واحد دائماً في الحالة غير العاكسة، أو صفر في الحالة العاكسة، وهذا هو الحال في أول دورتين في شكل ١٤-١٠. وأما في حالة وضع صفر في مسجل المقارنة فإنه ستكون هناك نبضة ضيقة جداً spike ستظهر على الطرف A. OC2A

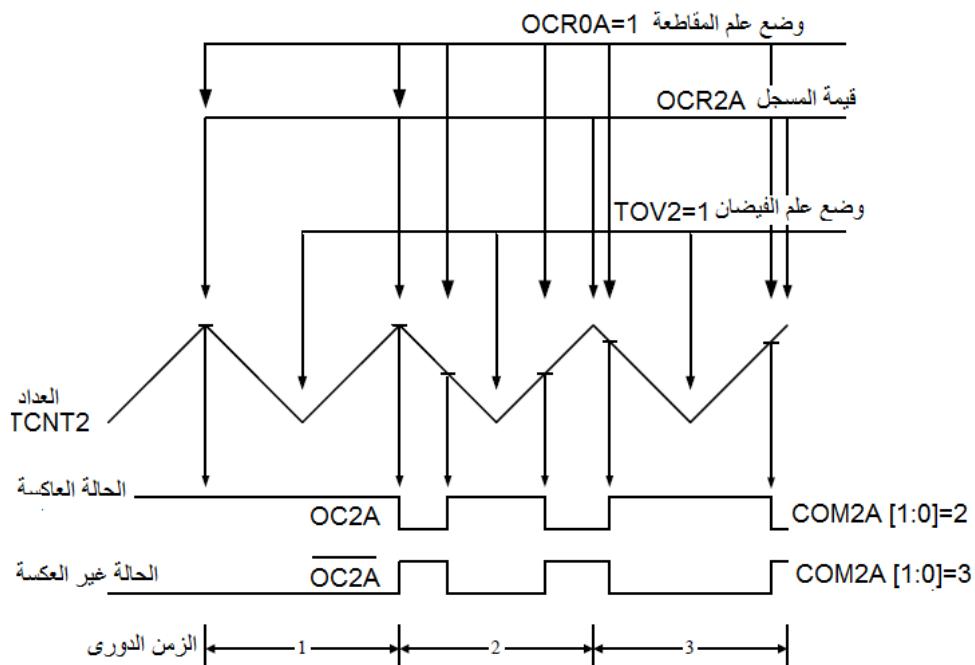
[يمكن تنفيذ برنامج تشغيل المؤقت ٢ في هذه الحالة \(تعديل النبضة السريع\) الموجود في الفصل ٨.](#)

#### ٤-طريقة تعديل عرض النبضة PWM المعدلة الطور phase correct

في هذه الطريقة تكون بتات توليد الشكل الموجي تساوى ١ (001) أو ٥ (101) كما في الجدول ٢-١٠. تتميز هذه الطريقة بأنها مزدوجة الميل، أحد الميلين يكون في حالة صعود العداد TCNT2 من صفر إلى القيمة العظمى TOP=0xFF، والثانى في حالة نزول العداد بالتدريج من القيمة العظمى للصفر. في حالة أن بتات الشكل الموجي WGM21، WGM20، و WGM22 تساوى ١ (001) تكون القيمة العظمى هي TOP=0xFF، وإذا كانت بتات الشكل الموجي تساوى ٥ (101)، فإن القيمة العظمى تكون TOP=OCR2A، أي أنها تكون القيمة المخزنة في المسجل OCR2A. مثل الطريقة السريعة السابقة ذات الميل الواحد، في حالة عدم العكس يتم تصفيير طرف خرج المقارنة OC2A عند تساوى العداد TCNT2 مع محتويات مسجل المقارنة OCR2A في حالة الصعود من الصفر إلى القيمة العظمى، ويتم وضعه بوحدة عند تساوى العداد TCNT2 مع محتويات مسجل المقارنة OCR2A في حالة النزول من القيمة العظمى للصفر كما في شكل ١٥-١٠. في حالة العكس يحدث عكس ذلك. بالطبع فإن طريقة الميل المزدوج سيكون أكبر تردد لها أقل من أكبر تردد في الطريقة السريعة ذات الميل الواحد. باختصار

وكما في شكل ١٥-١٠ الذي يبين التزامن في هذه الحالة، فإن العداد TCNT2 يعد صاعدا من الصفر إلى القيمة العظمى، ثم ينزل تدريجيا من القيمة العظمى للصفر مرة أخرى. عند التساوى مع مسجل المقارنة في أثناء الصعود، يصفر طرف خرج المقارنة OC2A، وعند التساوى مع مسجل المقارنة في أثناء النزول يضع الطرف OC2A بواحد مرة أخرى.

تذكر هنا أنه لكي تتم رؤية هذه الموجات على طرف المقارنة OC2A، فإن هذا الطرف لابد أن يكون قد تم تعينه ليكون طرف خرج بوضع وحيد في مسجل اتجاه البوابة التابع لها هذا الطرف كما أوضحتنا في شكل ١٣-١٠. لاحظ أنه في نهاية كل زمن دورى فإن علم الفيضان TOV2 يتم وضعه بواحد، وفي هذه الحالة إذا تم تنشيط كل من قناع المقاطعة TOIE2 وعلم المقاطعة العام I، فإنه يمكن القفز إلى برنامج خدمة مقاطعة ISR، ويمكن في هذا البرنامج تعديل قيمة مسجل المقارنة OCR2A على حسب الطلب.



شكل ١٥-١٠ التزامن الحادث مع طريقة تعديل عرض النبضة PWM المعدلة (الطور نسخة من دليل المتحكم)

تردد الموجة المعدلة العرض PWM ذات الطور المعدل يمكن الحصول عليها من المعادلة التالية:

$$f_{OC0APWM} = \frac{f_{clock}}{N \cdot 510}$$

حيث `fclock` هى نبضات تزامن المتحكم، و  $N$  هى نسبة القسمة المستخدمة للحصول على نبضات التزامن التى سيعمل عندها المؤقت (١ أو ٨ أو ٣٢ أو ٦٤ أو ١٢٨ أو ٢٥٦ أو ١٠٢٤). يمكن تنفيذ برنامج تشغيل المؤقت ٢ في هذه الحالة (تعديل النبضة المعدل الطور) الموجود في الفصل ٨.

## ٩-١٠ التشغيل غير المتزامن للمؤقت ٢

كما لاحظنا في كل ما مضى من هذا الفصل أنه ليس هناك أى فرق بين المؤقت ٢ والمؤقت صفر الذى سبق شرحه في الفصل ٨. وهذا فعلا هو الواقع، إنه طالما أن المؤقت ٢ يعمل في الحالة المتزامنة فإنه سيعتبر نسخة من المؤقت صفر. المقصود بالتشغيل المتزامن أن المؤقت يكون متزامنا مع نبضات تزامن المتحكم الأساسية `clk` سواء الداخلية من داخل المتحكم أو من خارجه. الجديد في المؤقت ٢ أنه يمكنه العمل مع نبضات خارجية يتم إدخالها على الطرفين ٩ و ١٠ لشريحة المتحكم باسم هذين الطرفين هو `TOSC1` على الطرف ٩ و `TOSC2` على الطرف ١٠، حيث يوضع بينهما طرف المذبذب الممثل لمصدر هذه النبضات. التشغيل غير المتزامن للمؤقت ٢ يعني أن هذه النبضات ستكون هي نبضات التزامن له وسيعمل عليها دون الحاجة أن يكون هناك مجھود لحاذة حواف (تزامن) هذه النبضات مع نبضات تزامن المتحكم، ولكن سيعمل المؤقت في هذه الحالة بطريقة مستقلة تماما عن نبضات تزامن المتحكم. نتيجة العمل بهذه الطريقة غير المتزامنة، يجب أن يكون هناك حرص تام عند الكتابة في أى مسجل من مسجلات المتحكم الخمسة وهى مسجل العد `TCNT2` ومسجل المقارنة `OCR2A` و `OCR2B` ومسجل التحكم في تشغيل المتحكم `TCCR2A` و `TCCR2B`، وإلا فمن الممكن أن يحدث خطأ في العداد وفي البيانات الموجودة في هذه المسجلات. لتجنب هذه الأخطاء الممكن حدوثها فإن كل واحد من هذه المسجلات الخمسة يكون له مسجل مؤقت مرفق له، وهذه المسجلات المؤقتة لا يتم تفعيلها إلا في حالة التشغيل غير المتزامن. أما في حالة التشغيل المتزامن فهذه المسجلات الخمسة تكون خاملة وليس لها أى استخدام.

في حالة التشغيل غير المتزامن، وعند الكتابة في أى مسجل من هذه المسجلات، فإن البيانات الجديدة يتم وضعها في المسجل المؤقت أولا، ويتم نقل هذه البيانات من المسجل المؤقت إلى المسجل الحقيقي أثناء فترات توقف المسجل الحقيقي. لذلك تم إضافة مسجل تحكم جدد يمكن من خلالهما معرفة إذا كان المسجل المراد الكتابة فيه نشطا أم لا ويتم أيضا من خلالهما التنقل بين التشغيل المتزامن وغير المتزامن، وسنقدم تفاصيل هذين المسجلين فيما يلى.

## مسجل التشغيل غير المتزامن ASR, ASSR

7	6	5	4	3	2	1	0
	EXCLK	AS2	TCN2UB	OCR2AUB	OCR2BUB	TCR2AUB	TCR2BUB
0	0	0	0	0	0	0	0

القيم التلقائية عند بداية التشغيل

شكل ١٦-١٠ مسجل التشغيل غير المتزامن في المؤقت ٢

شكل ١٦-١٠ يبين بثبات هذا المسجل حيث نلاحظ استخدام سبعة بثبات فقط منه والبت الثامنة غير مستخدمة. سنقدم هنا شرحاً لك كل واحدة من هذه البتات حيث من خلال هذا الشرح يمكننا أن نستزيد فهماً للتشغيل غير المتزامن.

**البت رقم ٦ تنشيط مصدر النبضات الخارجي EXCLK:** بوضع ١ في هذه البت، و اختيار النبضات غير المتزامن كمصدر لنبضات التزامن كما في شكل ٢-١٠ عن طريق وضع البت AS2=1، فإنه يتم تفعيل عازل النبضات الخارجية غير المتزامنة على الطرف ٩، TOSC1، للمتحكم لتصبح هي مصدر نبضات التزامن للمؤقت. تسجيل الواحد في هذه البت يجب أن يتم قبل اختيار التشغيل غير المتزامن.

**البت رقم ٥ العداد غير المتزامن AS2, AS2:** بوضع ١ في هذه البت يتم اختيار مصدر النبضات غير المتزامن على الطرف ٩، TOSC1 كما في شكل ٢-١٠. بوضع هذه البت تساوى صفر يتم اختيار المصدر المتزامن للنبضات وهي نبضات المتحكم clk. عند تغيير قيمة هذه البت، قد يحدث تغيير في القيم المسجلة في مسجلات المؤقت الخمسة السابق ذكرها.

**البت رقم ٤ علم اشغال العداد TCN2UB:** عندما يعمل المؤقت في الحالة غير التزامنية وتتم الكتابة في العداد TCNT2 فإنه أثناء الكتابة تصبح هذه البت تساوى واحد دلالة على أن العداد مشغول. بعد أن تتم الكتابة ويتم تجديد لقيمة العداد TCNT2 من المسجل المؤقت الخاص به، يتم تصفير لهذه البت، وهذا يدل على أن العداد TCNT2 جاهز لأى عملية كتابة أخرى.

**البت رقم ٣ علم انشغال مسجل المقارنة OCR2AUB:** عندما يعمل المؤقت في الحالة غير التزامنية وتم الكتابة في المسجل OCR2A فإنه أثناء الكتابة تصبح هذه البت تساوى واحد دلالة على أن هذا المسجل مشغول. بعد أن تتم الكتابة ويتم تحديد لقيمة المسجل OCR2A من المسجل المؤقت الخاص به، يتم تصفير لهذه البت، وهذا يدل على أن المسجل OCR2A جاهز لأى عملية كتابة أخرى.

**البت رقم ٤ علم انشغال مسجل المقارنة OCR2BUB:** عندما يعمل المؤقت في الحالة غير التزامنية وتم الكتابة في المسجل OCR2B فإنه أثناء الكتابة تصبح هذه البت تساوى واحد دلالة على أن هذا المسجل مشغول. بعد أن تتم الكتابة ويتم تحديد لقيمة المسجل OCR2B من المسجل المؤقت الخاص به، يتم تصفير لهذه البت، وهذا يدل على أن المسجل OCR2B جاهز لأى عملية كتابة أخرى.

**البت رقم ٥ علم انشغال مسجل التحكم TCR2AUB:** عندما يعمل المؤقت في الحالة غير التزامنية وتم الكتابة في المسجل TCR2AUB فإنه أثناء الكتابة تصبح هذه البت تساوى واحد دلالة على أن هذا المسجل مشغول. بعد أن تتم الكتابة ويتم تحديد لقيمة المسجل TCR2AUB من المسجل المؤقت الخاص به، يتم تصفير لهذه البت، وهذا يدل على أن المسجل TCR2AUB جاهز لأى عملية كتابة أخرى.

**البت رقم ٦ علم انشغال مسجل التحكم TCR2BUB:** عندما يعمل المؤقت في الحالة غير التزامنية وتم الكتابة في المسجل TCR2BUB فإنه أثناء الكتابة تصبح هذه البت تساوى واحد دلالة على أن هذا المسجل مشغول. بعد أن تتم الكتابة ويتم تحديد لقيمة المسجل TCR2BUB من المسجل المؤقت الخاص به، يتم تصفير لهذه البت، وهذا يدل على أن المسجل TCR2BUB جاهز لأى عملية كتابة أخرى.

إذا تمت الكتابة في أى مسجل من هذه المسجلات الخمسة أثناء انشغال هذا المسجل (بت الانشغال تساوى ١) فإنه من الممكن أن يحدث أخطاء غير معروفة في محتويات هذا المسجل ومن الممكن أن تحدث مقاطعات غير معروفة.

## مسجل التحكم العام في المؤقت General Counter Control Register, GTCCR

هذا المسجل ليس مخصصاً فقط للمؤقت ٢، ولكنه يتحكم في تزامن عداد معامل القسمة في المؤقتات الثلاثة، المؤقت صفر، المؤقت ١، والمؤقت ٢. كما نلاحظ في شكل ١٧-١٠ فإن هذا المسجل يتكون من ٨ بت مستخدم منها ثلاثة فقط تفاصيلها كالتالي:

7	6	5	4	3	2	1	0
TSM	0	0	0	0	0	PSRASY	PSRSYNC
القيم التلقائية عند بداية التشغيل							

شكل ١٧ مسجل التحكم العام في المؤقت ٢

**البت رقم ٠ إعادة وضع مقسم التردد للمؤقت صفر و ١ Prescaler Reset, PSRSYNC**: عندما تكون هذه البت تساوى ١ سيحدث إعادة وضع مقسم تردد المؤقت صفر والمؤقت ١، وكما نعلم أن كل من هذين المؤقتين يستخدمان نفس مقسم التردد كما رأينا في الفصل ٨ و الفصل ٩. عادة يتم تصفيير هذه البت فوراً عن طريق المتحكم إذا وجدت بواحد إلا إذا كانت البت TSM تساوى واحد.

**البت رقم ١ إعادة وضع مقسم التردد للمؤقت ٢ Prescaler Reset Timer2, PSRASY**: عندما تكون هذه البت تساوى ١ سيحدث إعادة وضع مقسم تردد المؤقت ٢. عادة يتم تصفيير هذه البت فوراً عن طريق المتحكم إذا وجدت بواحد إلا إذا كانت البت TSM تساوى واحد فإنه لن يتم تصفييرها. إذا تم وضع واحد في هذه البت بينما يعمل المؤقت ٢ في الحالة غير المتزامنة، فإنهما ستظل بواحد إلى أن يتم تصفيير عداد تقسيم التردد.

**البت رقم ٧ تنشيط وضع التزامن للمؤقتات Timer/Counter Synchronization mode, TSM**: عند وضع واحد في هذه البت فإن المؤقتات الثلاثة تدخل في وضع التزامن. في وضع التزامن يتم توقف عدادات التوقيت الثلاثة حتى يمكن إعادة تشكيل عداد مقسم التردد حتى نضمن عدم تقدم العدادات أثناء إعادة تشكيل مقسمات التردد. عند تصفيير البت TSM يحدث تصفيير فوري للبتات PSRASY و PSRSYNC وتبدأ عدادات المؤقتات في العد في نفس الوقت.

في نهاية هذا الفصل نؤكد أن المؤقت ٢ يشبه تماماً المؤقت صفر من حيث التركيب والتشغيل سوى عند تشغيل المؤقت ٢ في الحالة غير المتزامنة. فإنه في هذه الحالة سيكون مصدر نبضات التزامن له هو المذبذب غير المتزامن الذي يمكن توصيله على الطرف ٩ للمتحكم، وفي هذه الحالة يتم التحكم في أداء المتحكم من خلال مسجل التشغيل غير المتزامن ASSR الذي تم شرح وظيفته جميع بقائه.

## ملخص الفصل

كما ذكرنا في نهاية هذا الفصل فإن هذا المؤقت يشبه تماماً للمؤقت صفر سوى في حالة الاستخدام غير المتزامن. ولذلك إذا لم يتم استخدام المؤقت ٢ في الحالة غير المتزامنة فإنه ينصح باستخدام المؤقت صفر أو المؤقت ٢ أو كليهما معاً إذا كانت هناك حاجة لاستخدام أكثر من مؤقت في نفس الوقت.

# الفصل ١١



## المشغلات

## Actuators

**العناوين المضيئة في هذا الفصل:**

- ١ - مواتير التيار المستمر
- ٢ - مواتير المؤازرة
- ٣ - مواتير الخطوة

## ١-١١ مقدمة

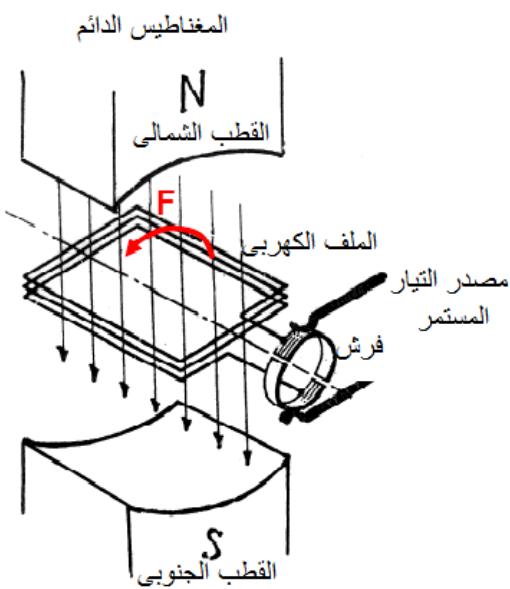
**لقد** رأينا في الفصول السابقة كيفية برمجة الملحقات المتاحة في المتحكم atmega328 وأهمها بوابات إدخال وإخراج البيانات، والمقاطعة، والمحول التماضي الرقمي، والمؤقتات الثلاثة المتاحة بداخله، وكتبنا برامج تطبيقية على كل من هذه المواضيع. من أهم التطبيقات على ذلك تشغيل المواتير بأنواعها، مواتير التيار المستمر dc motors، ومواتير المؤازرة stepper motors والتي أحياناً ستنطلق عليها مصطلح مواتير السيرفو، ومواتير الخطوة servo motors. سيكون الهدف من هذا الفصل هو التركيز على هذه المواتير الثلاثة من حيث التركيب وكيفية البرمجة وتطبيقات كل منها. المشغل actuator، هو أي نوع من أنواع المواتير يستخدم لتشغيل أو التحكم في آلية معينة أو نظام. هذا المشغل يعمل من خلال مصدر للطاقة، وهذا المصدر قد يكون مصدر كهربائي، أو هيدروليكي (يعمل بضغط الزيت عادة)، أو نيوماتي (يعمل بضغط الهواء). سنركز في هذا الفصل على المشغلات التي تدار كهربياً، وهي المواتير الثلاثة السابقات ذكرها. تستخدم المشغلات أينما تكون هناك حاجة للحركة أو الأتمتة كما هو الحال في الروبوتات، والتحكم في سivor نقل الحركة، وغير ذلك الكثير.

تستخدم المواتير الكهربائية في التطبيقات التي تتطلب دقة عالية في عدد اللفات أو حتى في الجزء من اللفة أو الحركات الترددية، فكل هذه التطبيقات يتم التعامل معها باستخدام المواتير الكهربائية، وهذا ما جعلنا نركز عليها فقط في هذا الفصل. تخيل الدقة المطلوبة من روبوت تكون مهمته مثلاً هي التقاط جزء معين من مكان ووضعه في مكان آخر، والمكائن محددين بدقة لا تتجاوز مثلاً نصف سنتيمتر. أو مثلاً تخيل روبوت يقوم بتنقيب الكارت الإلكتروني حيث تكون المسافات بين الثقوب وبعضها في حدود الميلليمتر أو أقل، كل هذه التطبيقات لا تجد في لها الآليات الهيدروليكيية أو النيوماتية.

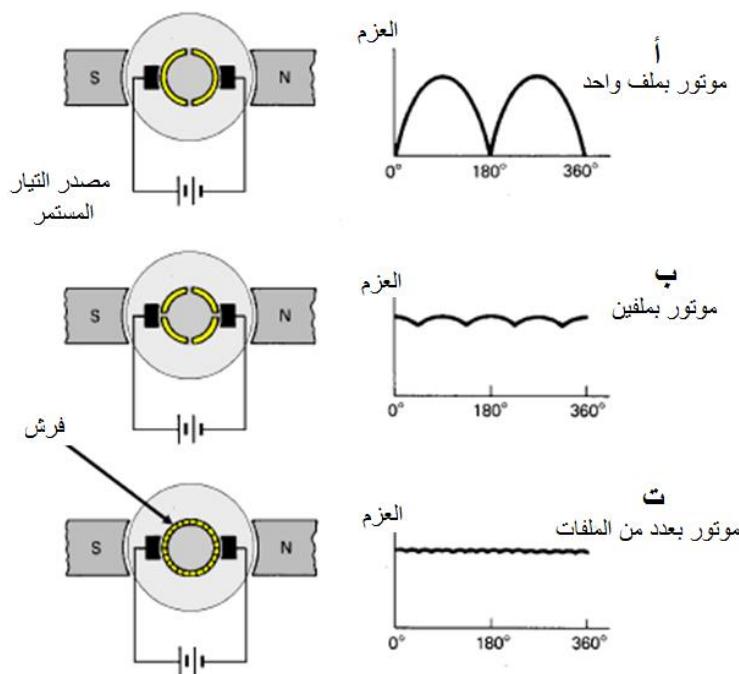
سنشرح هنا كل نوع من أنواع المواتير الثلاثة من حيث طريقة العمل والاختلاف بينه وبين الأنواع الأخرى وتطبيقاته ثم نقدم برامج تطبيقية على إدارة كل نوع باستخدام المتحكم atmega328. إننا لن نخوض في الكثير من الدراسة التصميمية لهذه الأنواع لأنه في هذه الحالة ربما تحتاج لكتاب مخصوص لذلك.

## ٢-١١ مواتير التيار المستمر DC motors

تقوم فكرة مotor التيار المستمر على أن تمرين التيار الكهربائي المستمر في ملف موجود في المجال المغناطيسي لمغناطيس دائم ينتج عنه دوران الملف الكهربائي. شكل ١-١١ يبين رسمًا تخطيطياً لهذه الظاهرة. الملف الكهربائي يمثل قلب المotor



شكل ١-١١ رسم تخطيطي لتركيب موتور التيار المستمر



شكل ٢-١١ تعليم حركة المotor بزيادة عدد الملفات

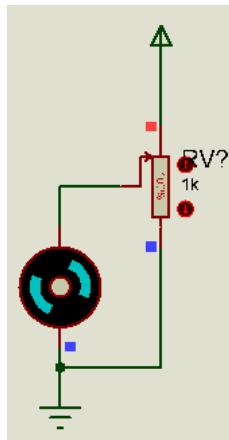
أو الجزء الدوار، والمغناطيس الدائم يمثل جسم المotor أو الجزء الثابت. اتجاه الدوران وسرعة الدوران يتم التحكم فيما عن طريق التيار الكهربائي المار في الملف. المشكلة هنا هي كيف سيتم إدخال التيار إلى الملف أثناء دورانه؟ يتم ذلك من خلال ما يسمى بالفرش brushes (مفرد فرشاة). الفرشاة تكون عادة من الكربون ومثبتة على المحور الدوار ويتم توصيل طرق الملف الدوار بها كما في شكل ١-١١. مصدر التيار يتم توصيله بهذه الفرش من خلال نقاط تلامس يتنتقل منها التيار إلى الملف أثناء دورانه. لذلك فإن عيوب هذا النوع من المواتير هو أن هذه الفرش ونقاط التلامس تنتج عنها شرارة كهربائية أثناء دوران الملف بسبب التلامس المتقطع بين الفرش ومصدر التيار المستمر. وهناك عيب آخر وهو تأكل هذه الفرش بسبب هذه الشرارة الناتجة بحيث تكون هناك حاجة لتغييرها من وقت لآخر. هذه العملية (ربط مصدر التيار المستمر مع الملف الدوار) تسمى عملية التبديل commutation، حيث يتم إبدال عملية إدخال التيار بين طرق الملف. نتيجة التوصيل بهذه الطريقة فإن التيار سيمر من أحد طرق الملف وبخرج من الطرف الآخر في نصف دورة، وفي نصف الدورة التالية يمر التيار من الطرف الآخر للطرف الأول. في هذه الحالة توجد فرشتان فقط كما في شكل ١-١١. شكل العزم (أو القوة) الناتجة من وجود ملف واحد كما في شكل

هذا الشارة الناتجة بحيث تكون هناك حاجة لتغييرها من وقت لآخر. هذه العملية (ربط مصدر التيار المستمر مع الملف الدوار) تسمى عملية التبديل commutation، حيث يتم إبدال عملية إدخال التيار بين طرق الملف. نتيجة التوصيل بهذه الطريقة فإن التيار سيمر من أحد طرق الملف وبخرج من الطرف الآخر في نصف دورة، وفي نصف الدورة التالية يمر التيار من الطرف الآخر للطرف الأول. في هذه الحالة توجد فرشتان فقط كما في شكل ١-١١. شكل العزم (أو القوة) الناتجة من وجود ملف واحد كما في شكل

١-١١ مع دوران الملف دورة كاملة ستكون كما في شكل ٣-١١. أى أن العزم سيكون قوى عندما يكون مسطح الملف متعمد مع المجال المغناطيسي بين القطبين وأقل ما يمكن عندما يكون مسطح الملف موازى للمجال المغناطيسي. لذلك للحصول على عزم ثابت بقدر الإمكان فإنه يتم استخدام أكثر من ملف واحد كما في شكل ٣-١١، وهذا هو الواقع في موافير التيار المستمر الحقيقية حتى تكون الحركة أكثر استمرارية ونعومة.

عادة يتم استخدام مغناطيس كهربى بدلاً من المغناطيس الدائم المستخدم في الجزء الثابت من المотор. في هذه الحالة يتم تغير تيار مستمر في ملف حول قلب حديدى ثابت حيث في هذه الحالة يصبح القلب الحديدى مغناطيساً. بذلك يصبح المotor به ملفات خاصة بالمجال المغناطيسي، وتسمى ملفات المجال، بجانب ملفات الجزء الدوار. في العادة يتم استخدام مصدر واحد للطاقة المستمرة لكل من الملفين. في هذه الحالة يتم توصيل الملفات الثابتة والمحركة إما على التوالى أو على التوازى، وكل طريقة من هاتين الطريقتين لها عيوبها ومميزاتها التي لن نخوض فيها. عموماً السمات العامة لموافير التيار المستمر يمكن تلخيصها فيما يلى:

- عند توصيل القدرة للمotor فإنه يدور في اتجاه معين بسرعة ثابتة، وعند عكس اتجاه التيار ينعكس اتجاه الحركة.
- يمكن استخدام النبضات المعدلة العرض PWM للتحكم في سرعة المotor، حيث ستتناسب سرعة المotor مع التيار المتوسط لهذه النبضات. بالطبع فإن قيمة هذا التيار ستتغير بتغيير مقدار عرض الموجة.
- هذا النوع من المواتير مناسب جداً مع السرعات العالية، ويستخدم في الكثير من التطبيقات بدءاً من لعب الأطفال حتى السيارات والقطارات الكهربائية.



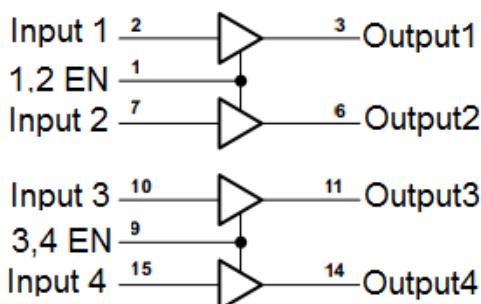
شكل ٣-١١

- رخيص الثمن بالنسبة لأنواع الأخرى من المواتير.
- مناسب للدوران المستمر مثل السيارات الكهربائية والقطار الكهربائي، والدوران الترددى، بمعنى أن المotor يدور عدد من اللفات في اتجاه معين ونفس العدد في الاتجاه الآخر.
- غير مناسب لتطبيقات اللف لجزء من الدورة، بمعنى يدور ٢٥ درجة مثلاً ويتوقف. يمكن استخدامه في هذا التطبيق بإضافة دوائر تغذية مرتبطة توقف المotor عند الوضع المطلوب، وهذه هي وظيفة مotor المؤازرة أو السيرفو كما سنترى.

فيما يلى سندرج في اللعب مع مotor التيار المستمر على برنامج البروتس. شكل ٣-١١ يبين إدارة مotor التيار المستمر مباشرةً من مصدر قدرة ومقاومة متغيرة للتحكم في سرعته.

الآن سنستخدم المتحكم atmega328 لإدارة اثنين موتور تيار مستمر من خلال شريحة الدفع L293D. عادة عند تشغيل المواتير تحتاج لدفع تيار لكي يوفر تيار عالي للمواتير ويكون بمثابة واجهة بين المتحكم والمواتير لسبعين مهمين: ١) أنه من الممكن استخدام جهد عالي (أكبر من ٥ فولت) لإدارة المотор بسرعة عالية دون توصيل هذا الجهد بالمحكم. ٢) أنه مع تشغيل المواتير فإنه قد ينشأ عنها نتوءات جهدية spikes قد تؤثر على عمل المحكم، هذه الشريحة تعمل على التخلص من هذه النتوءات الجهدية وتنعيمها. شكل ٤-١١ يبين الرسم الطرفي لهذه الشريحة حيث نلاحظ أنها يمكن بها تشغيل موتورين، كما أن بها ١٦ طرفاً وظائفها كالتالي:

- الطرف ١٦ وهو طرف القدرة الخاص بالدوائر الداخلية في الشريحة وهو يساوى ٥ فولت.
- الطرف ٨ وهو جهد القدرة المستخدمة في تشغيل المواتير وهذا الجهد يتراوح من ٥ فولت حتى ٣٦ فولت.
- الأطراف ٤ و ٥ و ١٢ و ١٣ كلها أطراف أرضي.
- المotor الأول M1 يتم توصيله بين الطرفين ٣ (output1)، والطرف ٦ (output2).

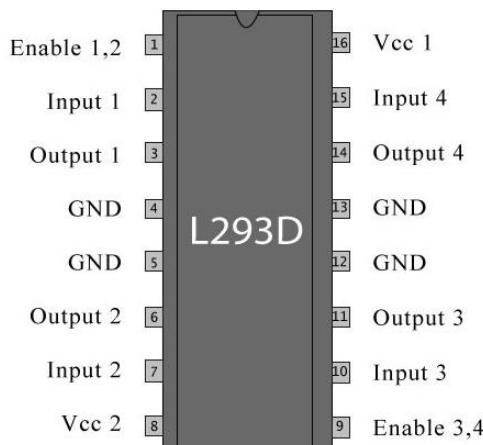


شكل ٤-١١ تنشيط أطراف كل من الموتورين

- الدخل للمotor الأول M1 يوضع على الطرفين ٢ (Input1) و ٧ (Input2). وهذان الطرفان يصلان على أي طرف خرج من أي بوابة إخراج من المحكم بحيث إذا كان الطرف ٢ يساوى واحد والطرف ٧ يساوى صفر، فإن المotor M1 سيدور عكس عقارب الساعة. وإذا كان الطرف ٢ يساوى صفر والطرف ٧ يساوى واحد، فإن المotor M1 سيدور مع عقارب الساعة.
- المotor الثاني M2 يتم توصيله بين الطرفين ١١ (output3)، والطرف ١٤ (output4).

الدخل للمotor الأول M2 يوضع على الطرفين ١٠ (Input3) و ١٥ (Input4). وهذان الطرفان يصلان على أي طرف خرج من أي بوابة إخراج من المحكم بحيث إذا كان الطرف ١٠ يساوى واحد والطرف ١٥ يساوى صفر، فإن المotor M2 سيدور عكس عقارب الساعة. وإذا كان الطرف ١٠ يساوى صفر والطرف ١٥ يساوى واحد، فإن المotor M2 سيدور مع عقارب الساعة.

الطرف ١ طرف تنشيط لأطراف المotor M1، والطرف ٩ طرف تنشيط لأطراف المotor M2 كما هو موضح في شكل ٤-١١. في هذا الشكل نلاحظ أن وضع الطرف ١ يساوى واحد سيجعل المotor M1 نشطاً، ووضعه



شكل ١١-٥ الرسم الطرفي للشريحة L293D

بصفر سيحمد هذا المотор أي يوقفه عن الدوران. نفس الكلام صحيح عن الطرف ٩ الذي ينشط أو يوقف المотор. ولذلك يمكن توصيل هذين الطرفين على طرف خرج M2 من المتحكم بحيث يستخدمان بمثابة مفاتيح للموتورين.

- شكل ١١-٥ يبين التركيب الطرفي للشريحة L293D.

البرنامج التالي يشغل موتورين من أطراف البوابة C حيث تم استخدام الطرفين PC0 و PC1 كطرف دخل للمotor الأول M1، و PC4 كطرف تنشيط لنفس المotor. الأطراف PC3 و PC5 و PC2 ستقوم بنفس الدور للمotor الثاني M2. المotorان يدوران في عكس عقارب الساعة لمدة ثانية. ثم يعكسان ويدوران في اتجاه عقارب الساعة لمدة ثانية وهكذا.

```
/*
 * Driving dc motor2.c
 *
 * Created: 9/5/2017 1:43:10 PM
 * Author : Mohamed Eladawy
 */
#include <avr/io.h>
#define F_CPU 1000000
#include <util/delay.h>

int main(void)
{
    DDRC = 0x0F; // initialize port C
                // motor1 across PC0, PC1,motor2 across PC2, PC3

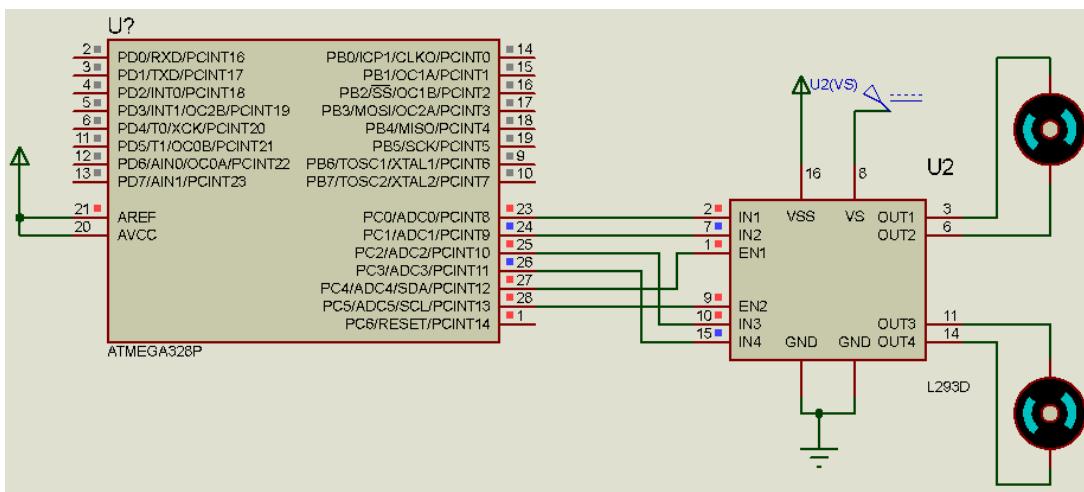
    while (1)
    { // clockwise rotation
```

```

PORTC = 0b00110101; // PC0 = High = Vcc
// PC1 = Low = 0
// PC2 = High = Vcc
// PC3 = Low = 0
_delay_ms(2000); // wait 1s
// counter-clockwise rotation
PORTC = 0b00111010; // PC0 = Low = 0
// PC1 = High = Vcc
// PC2 = Low = 0
// PC3 = High = Vcc
_delay_ms(2000); // wait 1s
}
}

```

شكل ٦-١١ يبين تنفيذ هذه الدائرة على البروتس. حاول تجربة أطراف التنشيط لكل من الموتورين.



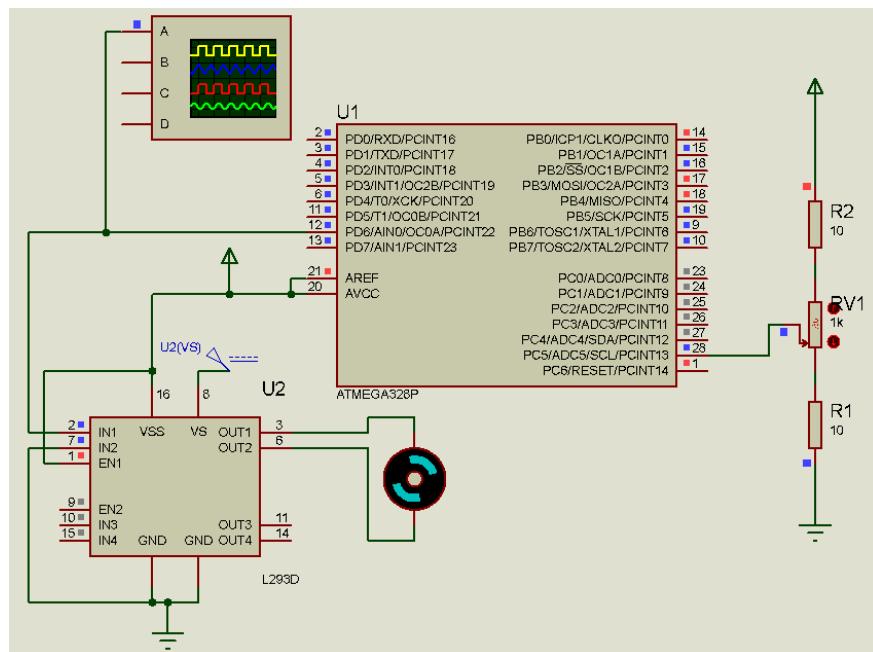
شكل ٦-١١ تشغيل موتورين تيار مستمر باستخدام المتحكم ودفع التيار L293D

يمكن التحكم في سرعة المOTOR باستخدام النبضات المعدلة العرض PWM. مثل هذه النبضات عندما يتم إدخالها على ملفات المOTOR فإن هذه الملفات تكون بمثابة مرشح تتعيّن للتيار الداخل بحيث تصبح سرعة المOTOR متناسبة مع متوسط الشكل الموجي المدخل. فعندما تكون الفترة التي تكون فيها الإشارة تساوي واحد on time كبيرة، فإن متوسط الموجة

يكون كبير، قريب من الواحد، وتكون سرعة المотор عالية. عندما يكون الفترة التي تكون فيها الإشارة تساوى صفر off time صغيرة، تكون القيمة المتوسطة صغيرة ومتقربة من الصفر، وبالتالي تقل سرعة المotor.

لقد سبق أن جربنا ذلك في الفصل ٨ في معرض الحديث عن المؤقت صفر، وكيفية الحصول منه على نبضات معدلة العرض واستخدامها في التحكم في سرعة مotor مستمر عن طريق مقاومة تكون هي بمثابة المتحكم في سرعة المotor.

لذلك يرجى الرجوع إلى الفصل ٨ وإعادة اللعب بهذا البرنامج، وربما يكون من المفيد استخدام الشريحة L293D كداعم للتيار بدلاً من الشريحة ULN2003A كما في شكل ٧-١١.



شكل ٧-١١ التحكم في سرعة مotor مستمر باستخدام النبضات المعدلة العرض PWM

### ١١-٣ موافير المؤازرة Servo motors

هذا النوع من الموافير عبارة عن مotor تيار مستمر مزود بصناديق تروس لتخفيض السرعة وحساس للموضع بحيث يمكنه استشعار الموضع وعمل تغذية مرتدية يتم من خلالها التحكم في سرعة دوران المotor والتحكم في موضعه بدقة. هذا النوع من الموافير له تطبيقات كثيرة منها الروبوتات وماكينات التحكم العددى CNC machines وغير ذلك الكثير. لذلك فإنه يمكن القول بأن مotor السيرفو يتكون من الأجزاء التالية:

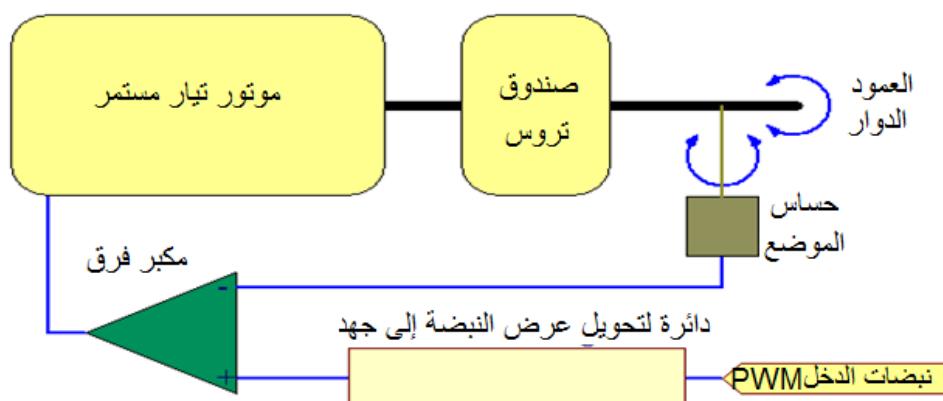
- ١ - مotor تيار مستمر

-٢ صندوق تروس لتخفيض السرعة

-٣ حساس للموضع على العمود الدوار للمotor

-٤ دائرة إلكترونية للتحكم في تشغيل المотор

شكل ٨-١١ يبين هذه الأجزاء وعلاقة كل منها بالآخر.

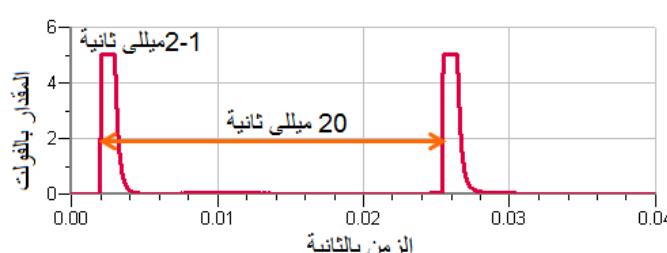


شكل ٨-١١ التركيب الداخلي لمotor المؤازرة

حساس الموضع في شكل ٨-١١ يقوم بقراءة زاوية العمود الدوار وتغذيتها إلى مكير الفرق. هذا الحساس يكون عادة عبارة عن مقسم جهد يعطي جهد خرج يتناسب مع زاوية دوران العمود. الطرف الآخر من مكير الفرق موصلاً على خرج دائرة إلكترونية تعطى جهذا يتناسب مع عرض النسبة الداخلية لها. خرج مكير الفرق وهو يكون سالب أو موجب على حسب مقدار أحد الدخلين بالنسبة للأخر يستخدم في إدارة المotor بحيث يقل هذا الفرق إلى الصفر دائماً، ويحدث ذلك عندما يصل العمود إلى الموضع المطلوب. وعلى ذلك فإن دائرة التحكم هذه تقرأ النسبات الداخلية إليها

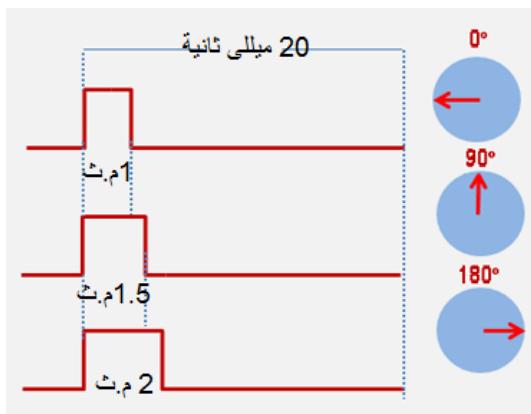
وبناء على عرض هذه النسبات تقوم بإدارة المotor في أحد الاتجاهين، إما مع أو عكس عقارب الساعة.

إن مواتير السيرفو لا تستجيب لأى نسبات مربعة، ولكن هذه النسبات يجب أن تكون محددة تماماً ومحسوبة بدقة. تقريباً، كل مواتير السيرفو يكون أقل عرض للنسبة فيها هو ١ ميليليانية، وأكبر عرض للنسبة هو ٢ ميليليانية، وإعطاء نسبات متتابعة للمotor أكبير من هذا العرض من الممكن أن يدمى دائرة التحكم ورما المotor بأكمله. شكل ٩-١١ يبين الشكل الموجي



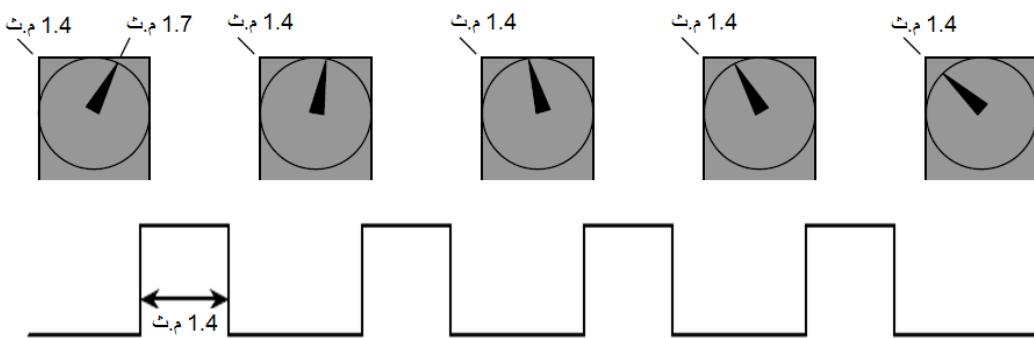
شكل ٩-١١ شكل النسبات اللازمة لتشغيل motor السيرفو

السيرفو يكون أقل عرض للنسبة فيها هو ١ ميليليانية، وأكبر عرض للنسبة هو ٢ ميليليانية، وإعطاء نسبات متتابعة للمotor أكبير من هذا العرض من الممكن أن يدمى دائرة التحكم ورما المotor بأكمله. شكل ٩-١١ يبين الشكل الموجي



شكل ١٠-١١ علاقة عرض النبضة بزاوية دوران المотор

شكل ١٠-١١ يوضح ذلك. معنى ذلك أن موتور المؤازرة يدور فقط في مدى ١٨٠ درجة، على حسب عرض نبضة التشغيل، وهذا هو الوضع التلقائي لهذا المotor. سری بعد قليل كيفية تشغيله ليدور دورات كاملة ومستمرة.



شكل ١١-١١ تحرك المotor من الوضع ١,٧ ميلي ثانية (الحالي) إلى الوضع ٤,١ ميلي ثانية (المطلوب)

شكل ١١-١١ يبين موتور مستقرا عند الوضع ١,٧ ميلي ثانية (يدين نقطة المركز)، والمطلوب تحريكه عقارب الساعة إلى الوضع ٤,١ ميلي ثانية (يسار نقطة المركز). في هذه الحالة يتم إدخال نبضات عرضها ٠,٤ ميلي ثانية (الوضع المطلوب) حيث نلاحظ أنه بعد عدد قليل من النبضات (٣ إلى ٥ نبضات) يصل المotor إلى الوضع المطلوب. بنفس الطريقة تتم الحركة إلى أي وضع. إذن الخلاصة من ذلك هي أنه بالنسبة لمotor السيرفو في شكله التقليدي (أحيانا يطلق عليه الشكل غير المعدل) فإن:

- إعطاء المotor نبضات عرضها ١ ميلي ثانية ستجعل المotor يدور عقارب الساعة ويستقر في أقصى اليسار (الوضع صفر درجة).

للنبضات المطلوبة لتشغيل هذا المotor، حيث نلاحظ أنها موجة معدلة العرض زمنها الدورى هو ٢٠ ميلي ثانية، والنسبة نفسها يتراوح عرضها من ١ إلى ٢ ميلي ثانية. عندما يكون عرض النبضة أقل ما يمكن (١ ميلي ثانية) يدور المotor إلى أقصى اليسار (الموضع صفر درجة)، وعندما يكون عرض النبضة يساوى ١,٥ ميلي ثانية يستقر المotor عند وضع المنتصف (٩٠ درجة)، وهذا الوضع يسمى عادة بالوضع المركب أو التلقائي. عندما يكون عرض النبضة ٢ ميلي ثانية يستقر المotor يدور فقط في مدى ١٨٠ درجة، على حسب عرض نبضة التشغيل، وهذا يوضح ذلك.

- إعطاء المотор نبضات عرضها ١,٥ ميللى ثانية ستجعل المotor يدور عكس عقارب الساعة (أو مع عقارب الساعة على حسب أين كان في الأصل) ويستقر في المنتصف تماماً، أو المركز (الوضع ٩٠ درجة).
- إعطاء المotor نبضات عرضها ٢ ميللى ثانية ستجعل المotor يدور مع عقارب الساعة ويستقر في أقصى اليمين (الوضع ١٨٠ درجة).

## الوضع المعدل والوضع غير المعدل لموتور المؤازرة

كل ما شرحناه مسبقاً كان عن ما يسمى بالوضع غير المعدل unmodified لموتور المؤازرة. وهو الوضع الذي يدور فيه المotor في مدى ١٨٠ درجة، وهو غالباً الوضع التقليدي الذي يتم شراء المotor عليه.

يمكن عمل تعديل بسيط على المotor غير المعدل وهو فصل دائرة التغذية المرتدة بحيث لا يتم قراءة زاوية دوران العمود الدوار. في هذه الحالة، وفي حالة عدم تطبيق أي نبضات دخل فإن المotor سيستقر عند المركز وستكون نقطة المركز هذه هي الوضع السابق الذي سنتم المقارنة به. الآن تخيل أننا وضعنا نبضات بعرض أكبر من ١,٥ ميللى ثانية. حيث عرض النبضة أكبر من ١,٥ ميللى ثانية، فإن المotor سيبدأ في التحرك عكس عقارب الساعة في اتجاه نقطة المركز، وحيث أن مسار التغذية المرتدة غير موجود، فإن قيمة الجهد الخارج من مكير الفرق سيظل هو المقابل لعرض النبضة الداخلية وسيستمر المotor في دورانه عكس عقارب الساعة ولن يتوقف إلا عند توقيف النبضات الداخلية. باختصار فإن المotor سيحاول الوصول إلى نقطة المركز التي لا يراها المotor لأن دائرة التغذية المرتدة مقطوعة.

بنفس الطريقة إذا وضعنا نبضات بعرض أقل من ١,٥ ميللى ثانية، فإن المotor سيدور في هذه الحالة في اتجاه عقارب الساعة محاولاً الوصول إلى نقطة المركز التي لن يراها لأن التغذية المرتدة مقطوعة، وبالتالي فإن المotor سيظل يدور في هذا الاتجاه إلى أن يتم إيقاف النبضات الداخلية. إذن الفرق بين المعدل وغير المعدل هو أن المotor غير المعدل له زاوية دوران مقدارها ١٨٠ درجة، ولا يستطيع أن يستمر في الدوران دورات كاملة. وأما المotor المعدل فهو نفس المotor غير المعدل مع حذف دائرة التغذية المرتدة حيث بناء على ذلك سيستطيع المotor الدوران في دورات كاملة وباستمرار.

وضع نبضات تساوى ١,٥ ميللى ثانية تماماً ستجعل المotor يتوقف تماماً ولن يتحرك.

سرعة المotor ستتوقف على مقدار بعد عرض النبضة الداخلية عن القيمة ١,٥ ميللى ثانية. وأما اتجاه الدوران فسيتحدد بناء على، هل هذه القيمة أكبر أم أقل من ١,٥ ميللى ثانية. شكل ١٢-١١ عبارة عن رسم توضيحي لذلك.

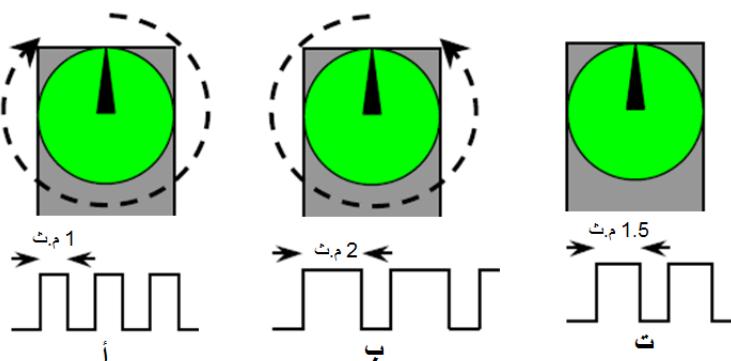
نلاحظ من هذا الشكل أن أي عرض للنبضة أقل من ١,٥ ميللى ثانية سيجعل المotor يدور مع عقارب الساعة كما في شكل ١٢-١١، وأى عرض للنبضة أكبر من ١,٥ ميللى ثانية سيجعل المotor يدور عكس عقارب الساعة كما في

نفس الشكل. وأما النبضات التي عرضها ١,٥ ميللي ثانية تماماً فسيجعل المотор يتوقف تماماً كما في شكل ١١-١٢.

عادة يخرج من هذا النوع من الموتير ثلاثة أسلاك بثلاثة ألوان. سلك باللون الأسود يتم توصيله على الأرضي، وسلك باللون الأحمر يتم توصيله بطرف القدرة (٥ فولت عادة)، والسلك الثالث يكون بلون أبيض أو أصفر أو أى لون مختلف عن الأسود والأحمر وهذا السلك توضع عليه إشارة النبضات المعدلة العرض.

### تشغيل موتور السيرفو باستخدام المتحكم atmega328

أولاً نزيد الحصول على تتابع من النبضات المعدلة العرض وتحقق الشروط التالية: الزمن الدورى (الزمن بين نبضتين متتاليتين) يساوى ٢٠ ميللى ثانية، وعرض النبضة يتراوح من ١ إلى ٢ ميللى ثانية كما أوضحها الشكل ٩-١١. للحصول على ذلك سنستخدم المؤقت ١ لكونه ٦ بت وبالتالي فإن أكبر قيمه يمكن أن يصل إليها ستكون ٦٥٥٣٦. سنستخدم هذا المؤقت في الحالة ١٤ (E) الموضحة في الجدول ٢-٩، حيث يتم وضع البتات  $WGM10=1$   $WGM13=WGM12=WGM11=1$  وهي حالة التعديل الموجى السريع حيث يقوم المؤقت بالبعد من الصفر إلى القيمة العظمى الموضوعة في المسجل ICR1A ثم ينزل للصفر وهكذا. سنجعل تردد نبضات التزامن للمتحكم هو ١٠٠٠٠٠٠ هرتز، وسنجعل معامل قسمة المؤقت يساوى ١ بحيث تكون نفس هذه النبضات هي نبضات تزامن المؤقت (أنظر جدول ٥-٩)، وبالتالي فإن الزمن الدورى للنبضة الواحدة سيكون ١٠٠٠٠ نبضة من هذه النبضات. وبالتالي فإننا سنجعل المسجل ICR1A=١٩٩٩٩ وهو ١٩٩٩٩.



شكل ١٢-١١ الدوران المستمر لمotor المقاورة المعدل

ناقص نبضة واحدة وهي النبضة رقم صفر. سنجعل الطرف OC1A وهو الخط OC1A صفر، ويظل كذلك إلى أن تصبح قيمة مسجل العد TCNT1 تساوى القيمة المخزنة في مسجل مقارنة OCR1A فيصبح واحد، ويظل واحد إلى أن يصبح العداد TCNT1 يساوى القيمة العظمى المخزنة في

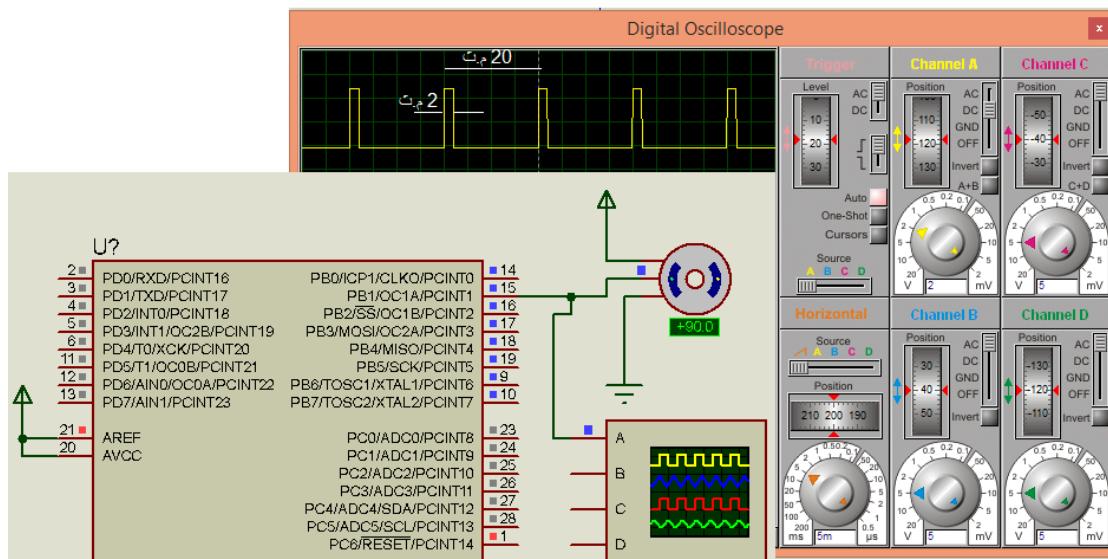
الخط ١٥ في شريحة المتحكم (PB1) يعمل في الحالة غير العاكسة (أنظر جدول ٣-٩). في هذه الحالة يكون الخط OC1A صفر، ويظل كذلك إلى أن تصبح قيمة مسجل العد TCNT1 تساوى القيمة المخزنة في مسجل مقارنة OCR1A فيصبح واحد، ويظل واحد إلى أن يصبح العداد TCNT1 يساوى القيمة العظمى المخزنة في

المسجل ICR1A فيصبح العداد صفر وينزل الخط OC1A إلى الصفر هو الآخر ليبدأ دورة جديدة. إذن معنى ذلك أن عرض النبضة (الزمن ON) سيكون هو الفترة ما بين صعود الطرف OC1A إلى الواحد، ثم نزوله للصفر بعد تساوى المقارنة بين العداد TCNT1 والمسجل OCR1A. نريد هذه الفترة أن تكون  $2 \text{ ميللي ثانية} = 2000 \text{ ميكروثانية}$ . لذلك سنضع القيمة  $18000 = 2000 - 2000$  في المسجل OCR1A. على هذا الأساس فإن البرنامج التالي سيقوم بهذه المهمة:

```
/*
* Servo motor1.c
*
* Created: 9/6/2017 8:51:33 PM
* Author : Mohamed Eladawy
*/
#define F_CPU 1000000
#include <avr/io.h>
int main(void)
{
    DDRB=0xFF; //port B output to enable the OC1A on PB1
    TCCR1A |= 1<<COM1A1 | 1<<COM1A0; // OC1A override PB1,
noninverting mode of OC1A
    TCCR1A |= 1<<WGM11; //Timer 1 mode E (14)
    TCCR1B |= 1<<WGM13 | 1<<WGM12 | 1<<CS10; // no prescalar
    ICR1 = 19999; //20000=20msec
    OCR1A = ICR1 - 2000; //2000=2msec

    while (1)
    {
    }
}
```

نتيجة تنفيذ هذا البرنامج موضحة في شكل ١٣-١١. حاول التأكد من تزامن الشكل الموجى الناتج على الأوسولوسكوب.



شكل ١٣-١١ نتائج تنفيذ برنامج موتور المؤازرة

نلاحظ في هذا الشكل أنه بمجرد تنفيذ البرنامج فإن المotor يتحرك ٩٠ درجة في اتجاه عقارب الساعة حيث أن عرض النبضة كان ٢ ميللي ثانية. ضع عرض النبضة يساوى ١ ميللى ثانية ( $OCR1A=ICR1=1000; // 1000 = 1msec$ ) حيث ستلاحظ أن المotor يتحرك بمقدار -٩٠ درجة في عكس اتجاه عقارب الساعة. ضع عرض النبضة يساوى ١,٥ ميللى ثانية ( $OCR1A=ICR1=1500; // 1500 = 1.5msec$ ) حيث ستلاحظ أن المotor لن يتحرك على الإطلاق لأن هذا هو وضع المركز.

بتتعديل بسيط في البرنامج السابق يمكن أن يجعل المotor يدور بحركة ترددية من اليسار لليمين ويمكن تنفيذه على نفس دائرة البرنامج السابق في البروتوس كما يلى:

```
/*
* Servo motor2.c
* Try on Proteus "Servo motor1"
* Created: 9/7/2017 7:38:25 AM
* Author : Mohamed Eladawy
*/
```

```

#define F_CPU 1000000
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
    DDRB=0xFF; //port B output to enable the OC1A on PB1
    TCCR1A |= 1<<COM1A1 | 1<<COM1A0; // OC1A override PB1,
noninverting mode of OC1A
    TCCR1A |= 1<<WGM11; //Timer 1 mode E (14)
    TCCR1B |= 1<<WGM13 | 1<<WGM12 | 1<<CS10; // no prescalar
    ICR1 = 19999; //20000=20msec
    OCR1A = ICR1 - 2000; //2000=2msec

while (1)
{
    OCR1A = ICR1 - 1000;
    _delay_ms(1000);
    OCR1A = ICR1 - 2000;
    _delay_ms(1000);
}
}

```

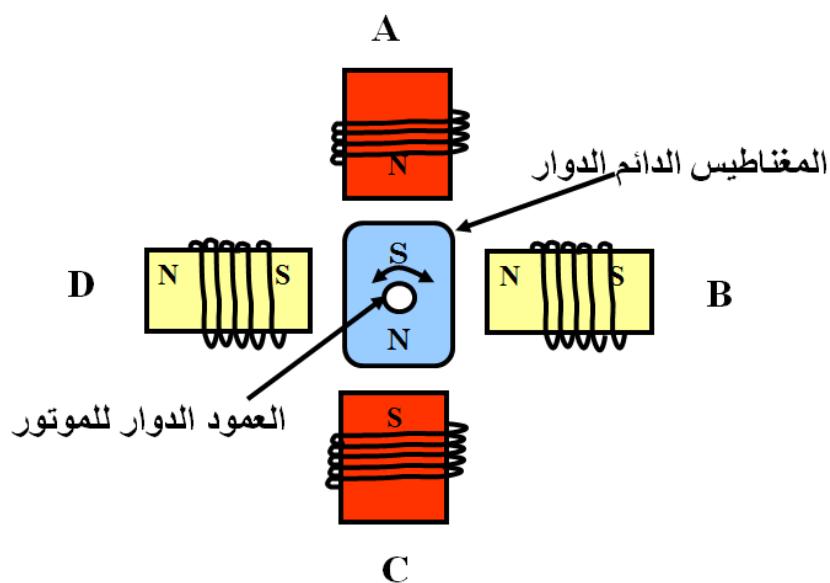
سنكتفى بهذا القدر عن موتور الموازرة. تذكر أنك إذا استخدمت موتور بقدرة عالية فإنك في الغالب ستحتاج لشريحة لدفع التيار مثل الشريحة L293D. بالطبع لم نستطع عمل تجربة على المotor المعدل لأن برنامج البروتوس لا يحتوى مثل هذا المotor، لأن عملية التعديل تقوم أنت بإجرائها، أو أنك تسؤال البائع أن يعطيك موتور معدل.

## ١١-٤ مواتير الخطوة Stepper motors

موتور الخطوة هو موتور تيار مستمر بدون فرش brushes أو نظام تبديل commutation وبالتالي فلن يكون هناك شرارة نتيجة الاحتكاك الميكانيكي مع الفرش، ولن يكون هناك تآكل للفرش بحيث تحتاج لتغييرها من فترة لأخرى. هذا

المotor يمكنه الدوران لجزء صغير من اللفة وبدقة عالية، ويمكن ضبطه ليدور عدد معين من الدورات بدقة عالية جداً، ويمكنه أن يدور وباستمرار في أي اتجاه تريده، وكل ذلك بدون الحاجة إلى دائرة تغذية عكسية كما هو الحال في مotor المقاورة السابق. هذا النوع من المواتير مناسب جداً لتشغيل الطابعات، والمسحات الضوئية scanners، وماكينات التحكم العددى CNC، وفي الكثير من التطبيقات التي تحتاج لدقة تشغيل عالية. من مميزات مotor الخطوة على باقي أنواع المmotورات أنه يمكن إدارته باستخدام إشارة رقمية ذات قدرة مناسبة. وعلى ذلك يمكن إدارته باستخدام معالج أو حاسوب بسهولة من خلال برنامج ينفذ بأى لغة من لغات البرمجة، ونحن سنرى كيفية إدارته باستخدام المتحكم .atmega328

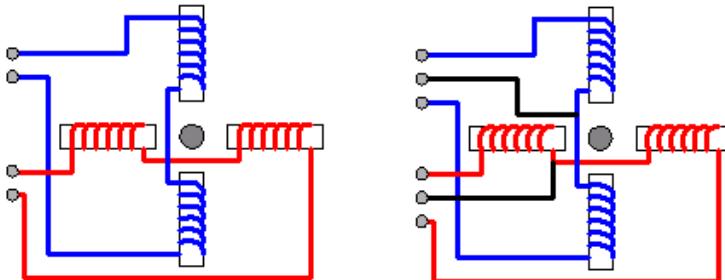
فكرة عمل مواتير الخطوة بأنواعها المختلفة كلها سهلة وبسيطة وسنقوم بتقديمها في هذا الجزء. شكل ١٤-١١ يبين التركيب الأساسي لمotor الخطوة . نلاحظ من هذا الشكل أنه يتكون من ٤ مغناطيسات كهربائية مثبتة stator وهي المغناطيسات A و B و C و D. يمرر التيار الكهربائي في الملف المحيط بأى واحد من هذه المغناطيسات يمكن مغنطة هذا المغناطيسي بحيث يكون أحد أطرافه هو القطب الشمالي والطرف الآخر هو القطب الجنوبي للمغناطيسي. يوجد في مركز المotor كما نرى في شكل ١٤-١١ مغناطيسي دائم حر الدوران حول مركزه الذي يمثل العمود الدوار للمotor . أي أنه بدوران هذا المغناطيسي يدور العمود وبالتالي يمكن إدارة الحمل الموصول على المotor.



شكل ١٤-١١ التركيب الداخلى لمotor الخطوة

تمرور التيار الكهربى في المغناطيسين A و C في الاتجاه المناسب فإنه يمكن جعل A قطب شمالى و C قطب جنوبى. ونتيجة وجودهما بجانب المغناطيس الدوار فإن المغناطيس الدوار سيعدل من وضعه بحيث يكون قطبه الجنوبي ناحية A وقطبه الشمالي ناحية C كما في شكل ١٤-١١ لأن الأقطاب المتشابهة تتناقض وال مختلف تتجاذب كما نعرف من قواعد

المغناطيسية البسيطة. أى أن الجزء الدوار سيعدل من وضعه بحيث يكون رأسيا كما في الشكل. إذا فصلنا التيار عن كل من A و C ، ووصلناه على كل من B و D فإن نفس الشيء سيحدث أيضا بحيث سيدور المغناطيس الدوار ليأخذ الوضع الأفقي في هذه المرة وسيكون اتجاه الدوران



شكل ١٥-١١ المواتير أحادية وثنائية القطبية

عكس أو مع عقارب الساعة على حسب اتجاه التيار في كل من B و D. فإذا كان D قطب شمالى و B قطب جنوبى فإن اتجاه الدوران سيكون عكس عقارب الساعة. بتوالى مرور التيار في الاتجاهات المناسبة في المغناطيسات الثابتة يمكن الاستمرار في دوران المغناطيس الدوار في الاتجاه المطلوب في خطوات كل منها ٩٠ درجة.

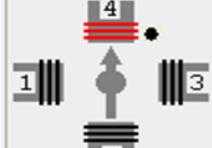
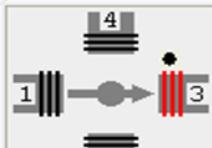
### المواتير أحادية وثنائية القطبية

هذا النوع من التغذية لملفات الجزء الثابت من المotor الذى شرحناه مسبقا يسمى التغذية ثنائية القطبية حيث يتم تغذية ملفين في نفس الوقت كما في شكل ١٥-١١. كما نرى من هذا الشكل فإن المotor ثنائى القطبية يخرج منه ٤ أسلاك، سلكان من كل ملفين حيث يتم تغذية الملفين مع بعضهما في كل مرة. وتكون التغذية بحيث يكون أحد الملفين قطب شمالى والأخر جنوبى لثبيت الجزء الدوار بينهما في وضع معين. هناك التغذية الأحادية، أو المotor أحادى القطبية الذى يحتوى أيضا ٤ ملفات ولكن يتم تغذية كل ملف على حدة ولذلك يخرج منه ٦ أسلاك حيث يكون الطرف الآخر للتغذية هو الأرضى. أحيانا يتم توصيل طرف الأرضى من داخل المotor بحيث يخرج من المotor ٥ أسلاك بدلا من ٦.

شكل ١٥-١١ ب يبين شكل توضيحي لملفات المotor أحادى القطبية.

شكل ١٦-١١ يبين شكلان توضيحا لطريقة دوران المotor تبعا لتتابع تغذية الملفات في مotor أحادى القطبية حيث كما نرى في هذا الشكل فإن مقدار الخطوة سيكون ربع لفة. شكل ١٧-١١ يبين طريقة أخرى لتغذية ملفات المotor

حيث يتم تغذية ملفين متجاورين في نفس الوقت فيبتعد عن ذلك وقوف الدوار بين الملفين وسيكون مقدار الخطوة ربع لفة أيضا في هذه الحالة.

Step	Coil 4	Coil 3	Coil 2	Coil 1	
a.1	on	off	off	off	
a.2	off	on	off	off	
a.3	off	off	on	off	
a.4	off	off	off	on	

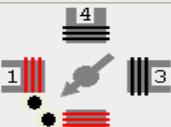
شكل ١٦-١١ تغذية موتور أحادى القطبية بحيث يكون مقدار الخطوة ربع لفة

يمكن إدارة المotor عن طريق تغذية ملف واحد حيث يقف الجزء الدوار في مواجهته، ثم تغذية نفس الملف والمجاور له فيقف الجزء الدوار بينهما، ثم تغذية الملف الأخير وحده فيقف الدوار في مواجهته، وهكذا حيث نرى أن خطوة الدوران في هذه الحالة ستكون ثمن لفة. شكل ١٨-١١ يبين رسم توضيحاً لذلك.

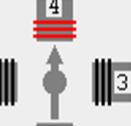
موتور الخطوة يمكن أن يوجد في السوق وله ٤ أطراف (ثنائي القطبية)، أو خمسة (أو ستة) أطراف (أحادى القطبية)، أو ثمانية أطراف (عام Universal) حيث يمكن توصيله خارجياً في أي من الحالتين السابقتين كما رأينا.

### مقدار خطوة المotor

يمكن زيادة عدد الخطوات في اللفة الواحدة في موتور الخطوة بطريقتين، الأولى وهي استخدام العديد من المغناطيسات الصغيرة في الجزيئين الدوار والثابت كما حدث في طريقة المغناطيسات الأربع السابقة. بالطبع هذه الطريقة ستعطى مواشير أكثر تعقيداً من حيث التركيب، ولكن طريقة التشغيل ستكون هي نفس الطريقة الأساسية السابقة.

Step	Coil 4	Coil 3	Coil 2	Coil 1	
b.1	on	on	off	off	
b.2	off	on	on	off	
b.3	off	off	on	on	
b.4	on	off	off	on	

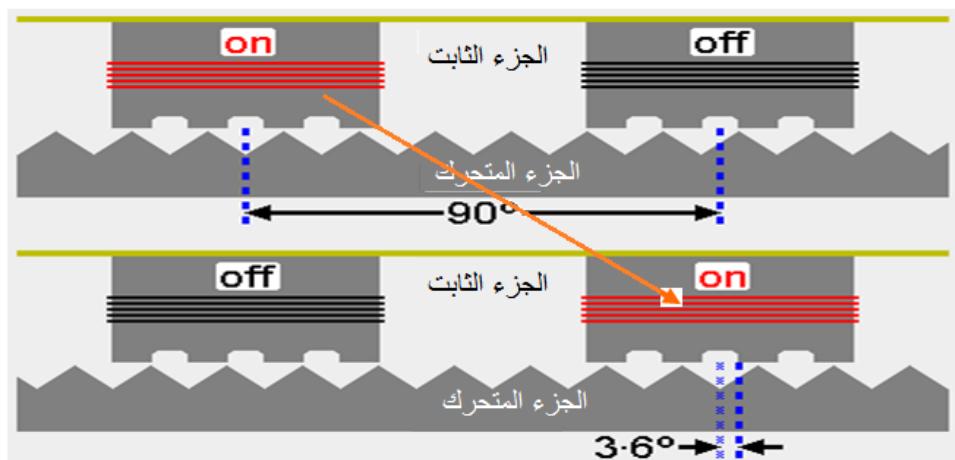
شكل ١٧-١١ تغذية ملفين متجاورين في نفس الوقت ، مقدار الخطوة ربع لفة أيضا

Step	Coil 4	Coil 3	Coil 2	Coil 1	
a.1	on	off	off	off	
b.1	on	on	off	off	
a.2	off	on	off	off	

شكل ١٨-١١ تغذية ملف، ثم ملفين، ثم ملف، وهكذا بحيث ينتج عن ذلك خطوة مقدارها ثمن لفة

يمكن زيادة عدد الخطوات في اللفة عن طريق عمل سنون في كل من المغناطيس الدوار والثابت كما في شكل ١١-١٩ . إذا كان عدد السنون في المغناطيس الدوار ٢٥ سنة وكل مغناطيس من الأربعة المثبتة به ٤ سنون، فإنه عند تغذية

أحد الملفات الثابتة تكون أسنان المغناطيس الدوار في منتصف أسنان المغناطيس الثابت ومقابلة لها. في نفس اللحظة تكون أسنان المغناطيس الدوار على حافة أسنان المغناطيس الثابت الذي عليه الدور في التغذية بمقدار ربع سنة من أسنان الجزء الثابت كما في الجزء العلوي من الشكل ١٩-١١. عند تغذية الملف الذي عليه الدور تتحرك أسنان المغناطيس الدوار بحيث تتطابق مع أسنان هذا الملف الثابت وفي منتصفها، ولذلك ستكون حركتها بمقدار ربع سنة فقط من أسنان الجزء الثابت . وعلى ذلك فإن مقدار الزاوية التي يتحركها المغناطيس الدوار ستكون  $360 \div (4 \times 25) = 3.6^\circ$  درجة. أي أنه سيكون هناك ١٠٠ خطوة في كل لفة.

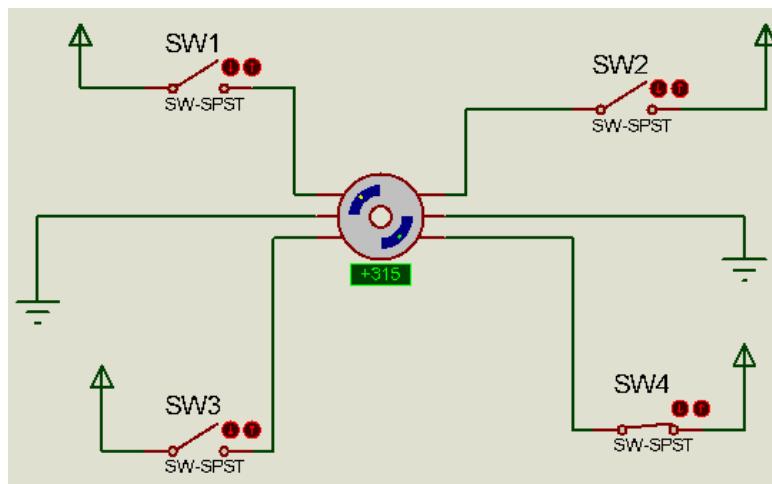


شكل ١٩-١١ عمل سنون في كل من الجزء الثابت والدوار للحصول على خطوة صغيرة

تتحدد سرعة المотор بسرعة تغذية هذه الملفات. كلما زادت هذه السرعة كلما كانت حركة المotor أكثر استمراً ونعومة، ومع تقليل هذه السرعة تكون حركة المotor في صورة خطوات منفصلة أيضاً. يجب ألا تزيد سرعة تغذية الملفات الثابتة عن حد معين بحيث يستطيع الجزء الدوار ملاحقة هذه السرعة. إذا زادت السرعة عن هذا الحد ربما تسمع زنة للمotor وقد يدور بسرعة أبطأ بكثير جداً من المتوقعة نتيجة الخطوات المفقودة لأن المotor غير قادر على ملاحقة تتبع تغذية الملفات.

## إدارة موتور الخطوة

هناك العديد من الدوائر والشرايع التي يمكن استخدامها في دفع التيار (إدارة) أو إدارة هذا النوع من المواتير. في هذا الجزء سنستخدم الشريحة L293D حيث قد سبق شرحها وتعودنا عليها في التعامل مع موتور التيار المستمر. ولكن قبل ذلك سنذهب لبرنامج البروتوس ونلعب قليلاً مع موتور باستخدام أربع مفاتيح كما في شكل ٢٠-١١.



شكل ٢٠-١١ إدارة موتور الخطوة من خلال أربع مفاتيح

باستخدام الأربع مفاتيح في شكل ٢٠-١١ يمكن أن تجرب دوران المотор بخطوات مقدارها ربع لفة أو ثمن لفة بالتتابعات التي شرحناها في الأشكال ١٥ و ١٦ و ١٧.

البرنامج التالي يبين استخدام المتحكم atmeg328 في إدارة موتور خطوة من خلال الشريحة L293D.

```
/*
* Stepper motor2.c
* Created: 9/7/2017 4:03:15 PM
* Author : Mohamed Eladawy
*/
#define F_CPU 1000000
#include <util/delay.h>
#include <avr/io.h>
int main(void)
{
    DDRB=0xFF; // port B output
    while (1)
    {
        PORTB=0x01; // activate first phase
```

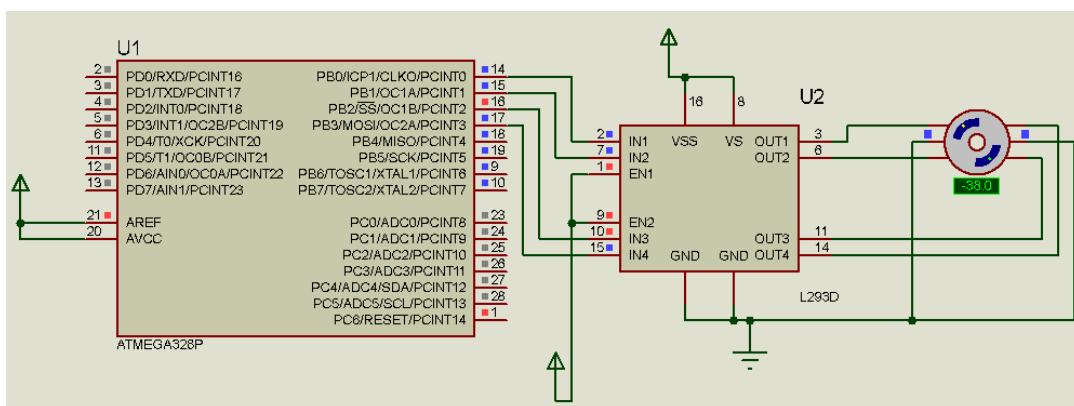
```

    _delay_ms(200);
    PORTB=0x02; // second phase
    _delay_ms(200);
    PORTB=0x04; // third phase
    _delay_ms(200);
    PORTB=0x08; // forth phase
    _delay_ms(200);
}

}
}

```

شكل ٢١-١١ يبين تنفيذ هذا البرنامج على البروتوس. لاحظ أن البرنامجعبارة عن حلقة مغلقة يتم فيها إخراج الأرقام 0x01 و 0x02 و 0x04 و 0x08 التي تغذي الملفات الأربعية على التوالي. حاول اللعب بهذا البرنامج لإدارة المотор في الاتجاهات المختلفة وبسرعات مختلفة.



شكل ٢١-١١ إدارة مotor الخطوة باستخدام المتحكم atmega328 عبر الشريحة L293D

## ملخص الفصل

بذلك تكون قد قدمنا في هذا الفصل شرحًا للموتورات الثلاثة من حيث نظرية عمل كل منهم وكيفية إدارتهم من خلال المتحكم atmega328 مع أمثلة وبرامج لكل نوع من الأنواع الثلاثة، بحيث يمكن للقارئ الاستعانة بهذه البرامج في تنفيذ أي تطبيق أو أي مشروع يستخدم هذه الموتورات.

## الفصل ١٢

### الاتصالات عبر واجهة الاتصالات المتتالية

#### Serial Peripheral Interface, SPI

العناوين المضيئة في هذا الفصل:

١- التراسل التتابعى فى المتحكم atmega328

٢- مسجل بيانات المسار SPI

٣- مسجل تحكم المسار SPI

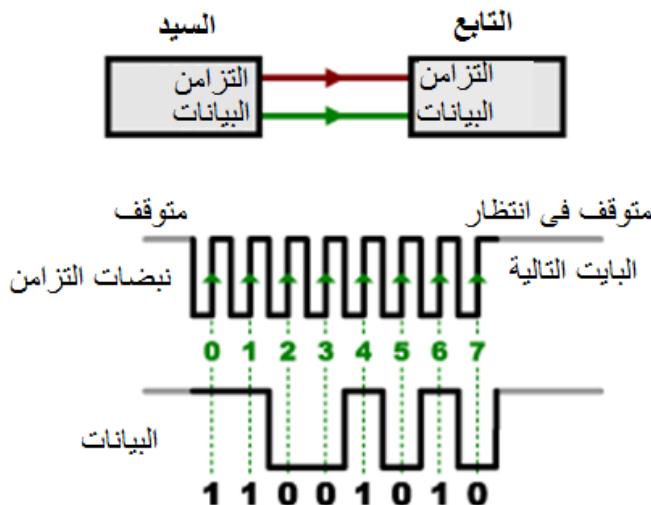
٤- مسجل حالة المسار SPI

٥- برمجة المسار SPI

## ١٢ - مقدمة

**تسمح** تقنية الاتصال المتتالي SPI (وتنطق اختصاراً سبي) بالاتصال السريع المتتالي والمترافق مع الأجهزة المحيطة بالمحكم أو بين محكم وآخر. المقصود بكلمة المتتالي هنا أن الاتصال بين المرسل والمستقبل يكون على سلك واحد ترسل عليه البيانات بت بعد بت، وكل بت ترسل متزامنة مع نبضة من نبضات التزامن (يرمز لها عادة بالرمز CLK أو SCK)، وهذا هو المقصود بكلمة المترافق هنا. لذلك ففي هذا النوع من الاتصالات يكون هناك سلك ثانٍ يربط ما بين المرسل والمستقبل توضع عليه نبضات التزامن التي ستتزامن معها البتات المرسلة على سلك البيانات. هذا على العكس من

نوع آخر من التراسل وهو التراسل غير المترافق Asynchronous communication حيث لا يلزم فيه أن يكون كل من المرسل والمستقبل متزامنين مع نفس نبضات التزامن ولذلك يوجد فيه خط لنقل نبضات التزامن بينهما وسيأتي شرح هذا النوع من التراسل في الفصل القادم.



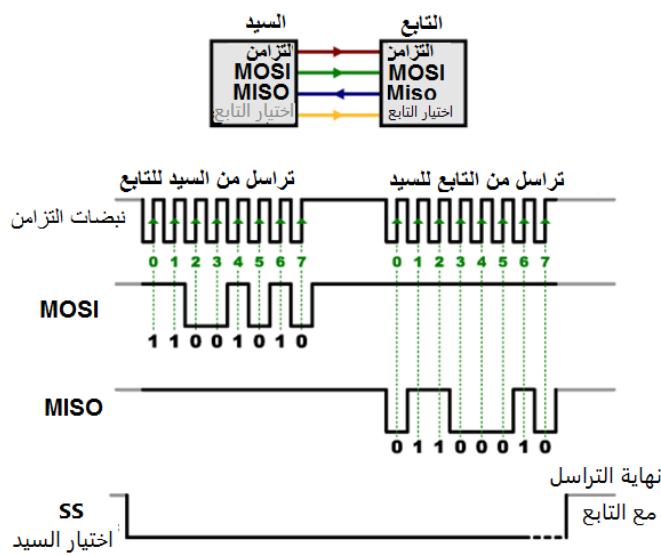
شكل ١-١٢ رسم تخطيطي لعملية التراسل المتتالي

شكل ١-١٢ يبين رسم تخطيطياً لعملية التراسل المتزامن حيث نلاحظ وجود خط لنقل التزامن بين كل من طرف التراسل وخط آخر للبيانات. طرف التراسل يسمى بالسيد أو الماستر master حيث

هو الذي تصدر من عنده نبضات التزامن وهو المسئول عن تحديد سرعتها. الطرف الثاني يسمى العبد أو التابع slave حيث يكون مستقبل فقط للبيانات والتزامن القادمين من الماستر وليس له أي دور في تحديدها. في شكل ١-١٢ البيانات المرسلة هي 11001010 ومع أول نبضة تزامن (النبضة رقم 0) يتم قراءة البت رقم صفر من البيانات (1) وإرسالها، ومع النبضة رقم 1 يتم قراءة البت الثانية من البيانات وهي (1 أيضاً) وإرسالها، ومع النبضة رقم 2 يتم قراءة البت الثالثة من البيانات وهي (0) وإرسالها، وهكذا حتى النبضة رقم 7 حيث يتم قراءة البت الثامنة من البيانات وهي (0) وإرسالها، وبذلك تنتهي عملية إرسال هذه البايت التي بدأت من البت ذات القيمة العظمى وانتهت بالبت ذات القيمة الصغرى في البيانات. عملية قراءة بت معينة من البيانات وإرسالها في شكل ١-١٢ تتم مع الحافة الصاعدة من

نبضات التزامن. بعض الأنظمة تعمل مع الحافة النازلة من نبضات التزامن. عادة في نظم التراسل التي من هذا النوع يكون هناك سيد أو ماستر واحد ومن الممكن أن يكون هناك تابع واحد أو عدة توابع يختار منها السيد واحد ليتم التراسل معه.

عملية التراسل مع نظام السبائ SPI تتم بطريقة الإزدواج الكامل full duplex بمعنى أن كل من طرف التراسل (السيد والتابع) يمكن أن يرسل كل منهما للآخر في نفس الوقت على خط بيانات منفصل، لذلك فإن هناك خط بيانات



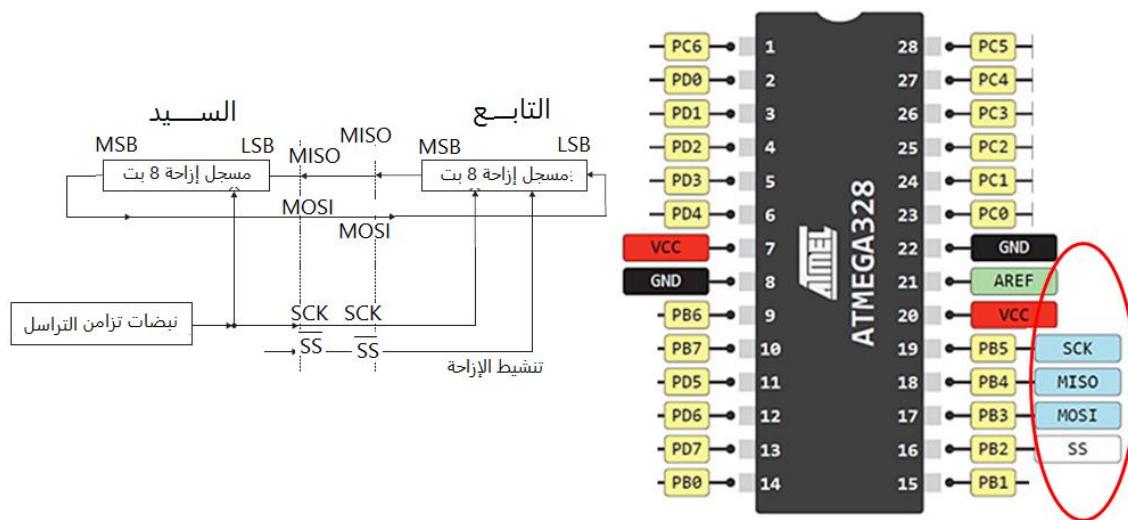
شكل ٢-١٢ التراسل المزدوج بين السيد والتابع والعكس

في شكل ٢-١٢ تم إرسال البيانات 11001010 على الخط MOSI من السيد للتابع، والبيانات 01100010 من التابع للسيد. لاحظ استخدام نفس نبضات التزامن الخارجية من السيد. لاحظ أيضاً استخدام إشارة اختيار التابع SS الخارجة من السيد، والتي بمجرد تنشيطها (نولها للصفر) يبدأ تنشيط التابع وتبدأ عملية التراسل في الاتجاهين. بانتهاء التراسل مع إرسال آخر بait يصعد خط اختيار التابع للواحد حيث يتم إخماد التابع بحيث لا يستجيب لأى تراسل.

ينقل البيانات من السيد للتابع، وخط آخر ينقل البيانات من التابع للسيد، وهذا بالإضافة طبعاً لخط نبضات التزامن CLK. عندما يقوم السيد بإرسال بيانات إلى التابع فإنه يرسلها على طرف Master out/Slave In, MOSI، بمعنى أن السيد يرسل (يخرج البيانات) والتابع يستقبل (يدخل البيانات). ولذلك يتم ربط MOSI في السيد مع الطرف في التابع. عندما يريد التابع أن يرسل بيانات إلى السيد، فإنها يرسلها على الطرف Master in/Slave out, MISO، بمعنى أن السيد

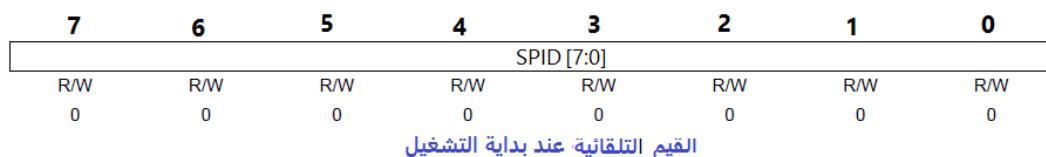
## ٢-١٢ التراسل التتابعى SPI في المتحكم atmega328

لقد رأينا في الجزء التقديمي السابق فكرة عامة عن التراسل التتابعى من خلال المسار SPI على وجه العموم، وهنا سنركز الحديث عن التراسل التتابعى في المتحكم atmega328 الذي هو موضوع دراستنا في هذا الفصل. التراسل في المتحكم لن يحيى عن الفكرة العامة التي ذكرناها مسبقاً. شكل ٣-١٢ يبين أطراف المتحكم المستخدمة في هذه العملية من التراسل حيث نلاحظ وجود نفس الأطراف الأربع وبنفس التسمية وهي الخطوط SCK الذي ستخرج عليه نبضات التزامن الخاصة بالتراسل، والخط MOSI، والخط MISO، وخط تنشيط التابع  $\overline{SS}$  المنخفض الفاعلية ولذلك تم وضع شرطة على إسمه. يمكن فهم ما تبقى من هذا النوع من التراسل من خلال شرح المسجلات المستخدمة في هذا النوع من التراسل.



شكل ٣-١٢ أطراف المتحكم atmega328 المستخدمة في المسار SPI وطريقة توصيل السيد والتتابع

## ٣-١٢ مسجل بيانات المسار SPI



شكل ٤-١٢ مسجل بيانات المسار SPI

مسجل بيانات التراسل المتتالي Serial Peripheral Data Register, SPDR يتكون من 8 بت كما في شكل ٤-١٢ . في حالة وضع المتحكم ليعمل على أنه السيد master، فإن البيانات المطلوب إرسالها تتابعاً يتم وضعها في هذا المسجل. خط اختيار التابع  $\overline{SS}$  يتم وضعه بصفر لتنشيط التابع. بعد ذلك، بمجرد كتابة بايت (8 بت) في المسجل SPDR، فإن مولد ذبذبات التزامن يبدأ في العمل، وتبدأ الإزاحة لمحتويات مسجل البيانات ناحية اليسار بحيث تخرج البت ذات القيمة العظمى MSB أولاً على الطرف MOSI متوجهة إلى المستقبل (التابع). بعد ثمانية نبضات تزامن، أي استكمال إزاحة محتويات مسجل البيانات بالكامل (1 بايت) تتوقف نبضات التزامن لتنتهي عملية إزاحة هذه البايت. في هذه الحالة يصبح علم مقاطعة نهاية الإرسال SPIF يساوى واحد. في هذه الحالة إذا كان علم تنشيط مقاطعة التراسل SPIE يساوى واحد، وكان علم المقاطعة العام I يساوى واحد أيضاً، فإن المتحكم سيقفز إلى برنامج لخدمة مقاطعة التراسل، وسترى ذلك بالتفصيل مع الحديث عن مسجل التحكم. يمكن الاستمرار في إرسال بايت آخرى بتحميلها في المسجل SPDR. لإنهاء عملية التراسل يجب على المستخدم أن يضع الخط  $\overline{SS}$  يساوى واحد لإنتهاء هذه الدورة من التراسل.

في حالة وضع المتحكم على أنه تابع Slave، فإنه سيظل خاماً إلى أن يتم تنشيط خط اختيار التابع  $\overline{SS}$  يجعله يساوى صفر، في هذا الحالة تبدأ البيانات في الدخول إلى مسجل البيانات بت وراء بت إلى أن يتم استكمال استقبال بايت كامله، حيث عندها تتوقف نبضات التزامن، ويصبح علم المقاطعة SPIF يساوى واحد، وإذا كان علم تنشيط المقاطعة SPIE نشطاً وعلم المقاطعة العام I يساوى واحد فإنه يمكن القفز إلى برنامج خدمة مقاطعة.

لاحظ أنه في حالة الإرسال من السيد فإنه يجب ألا تكتب بيانات جديدة في مسجل البيانات SPDR قبل الانتهاء من دورة الإزاحة (إزاحة بايت كاملة)، وفي جانب التابع فإنه يجب قراءة محتويات مسجل البيانات SPDR قبل الانتهاء من عملية الإزاحة التالية وإلا فإنها ستسجل على البيانات السابقة وبذلك فإن هذه البايت ستضيع.

## ٤-١٢ مسجل تحكم المسار SPI

7	6	5	4	3	2	1	0
SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
0	0	0	0	0	0	0	0

القيم التلقائية عند بداية التشغيل

شكل ٤-١٢ مسجل تحكم المسار SPI

مسجل تحكم المسار SPI (Serial Peripheral Control Register, SPCR) يتكون من 8 بت كما هو موضح في شكل ١٢-٥. تفاصيل وظيفة كل بت من هذه البتات ستكون كما يلى:

**البتات ٠ و ١ في المسجل SPCR والبت ٠ في المسجل SPISR:** هذه البتات الثلاثة SPR0 و SPR1 و SPI2X في مسجل التحكم SPCR والبت SPI2X في مسجل الحالة SPSR، الذي سيأتي شرحه بعد قليل، تتحكم في تردد نبضات تزامن المسار كنسبة من نبضات تزامن المتحكم كما في الجدول ١-١٢. لاحظ في الجدول أن هناك ثمانى قيم للتردد، في الأربعة الأولى تكون البت SPI2X=0، وفي الأربعة الثانية تكون البت SPI2X=1، وفي هذه الحالة يكون التردد ضعف المقابل له في الأربعة الأولى.

**البت ٢، CPHA:** هذه البت خاصية بطور نبضة التزامن Clock Phase، عند وضع هذه البت بوحدة فإن البيانات يتم إزاحتها مع الحافة المتقدمة من نبضات التزامن، وعند وضع البت CPHA=0 فإن الإزاحة تكون عند الحافة المتأخرة من نبضات التزامن.

**البت ٣، CPOL:** عند وضع هذه البت تساوى واحد CPOL=1، يكون طرف نبضات التزامن خاماً، وبوضع هذه البت تساوى صفر CPOL=0، يكون طرف نبضات التزامن خاماً.

جدول ١-١٢ نبضات تزامن المسار SPI نسبة لنبضات تزامن المتحكم (نسخة من دليل المتحكم)

SPI2X	SPR1	SPR0	تردد نزامن المسار SPI
0	0	0	fosc/4
0	0	1	fosc/16
0	1	0	fosc/64
0	1	1	fosc/128
1	0	0	fosc/2
1	0	1	fosc/8
1	1	0	fosc/32

**البت 4، MSTR:** هذه البت تختار بين حالي السيد والتابع Master/Slave، حيث يوضع هذه البت تساوى واحد سيجعل المتحكم يعمل كسيد، ووضعها تساوى صفر سيجعل المتحكم يعمل كتابع.

**البت 5، DORD:** يوضع هذه البت تساوى واحد فإنه سيتم إرسال البت ذات القيمة الصغرى LSB من البيانات أولاً، وبوضعها تساوى صفر يتم إرسال البت ذات القيمة الكبيرة MSB أولاً.

**البت 6، SPE:** هذه البت خاصة بتفعيل المسار المتتالى، فبوضعها تساوى 1 يصبح المسار SPI مفعلاً، وبوضعها تساوى صفر يصبح المسار SPI خاملاً. لذلك يجب وضعها تساوى واحد لتنفيذ أي عملية على هذا المسار.

**البت 7، SPIE:** هذه البت تمثل علم تنشيط المقاطعة الخاص بالمسار SPI، فإذا كان علم المقاطعة SPIE=1 فبوضع SPIF في مسجل الحالة SPSR يساوى واحد، وكان علم المقاطعة العام I في مسجل الحالة الخاص بالمحكم يساوى واحد، فإن المحكم سيقفز إلى برنامج خدمة مقاطعة ISR خاص بهذا المسار. بذلك تكون قد تعرفنا على وظائف جميع برات مسجل التحكم SPCR الخاص بالمسار SPI.

## ٥-١٢ مسجل حالة المسار SPI

شكل ٦-١٢ يبين برات مسجل الحالة SPSR حيث نلاحظ استخدام ثلاثة برات منه فقط.

7	6	5	4	3	2	1	0
SPIF	WCOL	0	0	0	0	0	SPI2X

القيم التلقائية عند بداية التشغيل

شكل ٦-١٢ مسجل الحالة SPSR في المسار SPI

**البت 0، SPI2X:** لقد رأينا أن وظيفة هذه البت هي تحديد تردد نبضات تزامن المسار كما في جدول ١ بالمشاركة مع البارات 0 و 1 في مسجل التحكم SPCR.

**البت 6، WCOL:** هذه البت تبين حدوث تصادم بيانات Write Collision flag، بمعنى كتابة بيانات في مسجل البيانات SPDR قبل الانتهاء من إزاحة البيانات الحالية خارج المتحكم. لاحظ أن المتحكم هو المسئول عن وضع هذه البت بوحدة عند حدوث هذا التصادم.

**البت 7، SPIF:** هذه البت Serial Peripheral Interrupt Flag تصبح واحداً (عن طريق المتحكم) عند استكمال إرسال بait كاملة. يمكن الدخول في برنامج خدمة المقاطعة إذا كان هذا العلم SPIF، وكان علم تنشيط المقاطعة SPIE في مسجل التحكم SPCR، وكان علم المقاطعة العام I في مسجل حالة المتحكم كلها تساوى واحد. بمجرد القفز إلى برنامج خدمة المقاطعة يعود العلم SPIF إلى الصفر مرة أخرى تمهيداً للدورة إرسال أخرى. بذلك تكون قد انتهينا من شرح بيات مسجل الحالة SPSR الخاص بالمسار SPI.

## ٦-١٢ برمجة المسار SPI

برمجة المسار SPI لإرسال واستقبال بيانات بين شريحة متحكم atmega328 تعمل كسيد أو ماستر، وشريحة متحكم مثلها تماماً Atmega328 تعمل كتابع أو سليف، تعتبر عملية سهلة جداً. بعد أن عرفنا وظيفة كل بيت من بيات مسجلات التحكم والحالة وكذلك وظيفة مسجل البيانات سنرى هنا كيف نبرمج اثنين متحكم ليقوما بهذه المهمة. إن فهم ما يحدث عند إرسال بait من السيد إلى التابع كما شرحناها في شكل ٣-١٢ ستسهل هذه المهمة بدرجة كبيرة. كما هو واضح في شكل ٣-١٢ فإنه عند إرسال بait من المرسل بت بعد بت فإن هذه البتات تزاح في مسجل بيانات المستقبل، وفي نفس الوقت فإن محتويات مسجل بيانات المستقبل تزاح هي الأخرى إلى مسجل بيانات المستقبل في صورة دائرة. أي أنه بعد إقامة عملية إرسال بait كاملة فإن محتويات مسجل بيانات كل من المرسل والممستقبل يتم استبدالها. ولذلك فإن عملية إرسال بait من المرسل (السيد) إلى التابع (الم المستقبل) أو العكس يمكن أن تتم بنفس مجموعة الأوامر كالتالي:

```
SPDR = data;  
while(!(SPSR & (1<<SPIF)));
```

حيث الأمر الأول يضع البيانات المطلوب إرسالها في مسجل البيانات، وبعدها مباشرة يبدأ المتحكم في إرسالها بت بعد إرسال آخر بت يتم وضع علم المقاطعة SPIF بوحدة وتتوقف عملية التراسل. لذلك فإن الأمر الثاني يتطلب

إلى أن يصبح علم المقاطعة يساوى واحد دلالة على انتهاء عملية التراسل. في هذه الحالة تصبح محتويات مسجل البيانات SPDR هى البيانات القادمة من الطرف الآخر. على ذلك يمكن كتابة هذين الأمرين عند المتحكم التابع وبعد إتمام عملية التراسل ستكون محتويات المسجل SPDR هى البيانات القادمة من المتحكم السيد. كما يمكن كتابة نفس الأمرين عند المتحكم السيد، وبعد إتمام عملية التراسل ستكون محتويات المسجل SPDR هى البيانات القادمة من المسجل التابع. المثال التالي سيوضح هذه العملية تماما.

## مثال

في هذا المثال سنستخدم اثنين متحكم atmega328 بحيث يعمل أحدهما متحكم سيد، والثانى سيكتب بمثابة متحكم تابع. سنوصل خطوط المسار SPI في المتحكمين مع بعضهما كما في شكل ٧-١٢. خط اختيار التابع  $\overline{SS}$  في المتحكم السيد سنضعه يساوى واحد (غير فعال) دائمًا لأنه السيد، ونفس الطرف في المتحكم التابع سنضعه يساوى صفر (الأرضى) حتى يتم اختياره كسيد دائمًا. سنبرمج عداد يعد من الصفر ويزداد بمقدار واحد بعد زمن تأخير معين ونرسل قيمة العداد من المتحكم السيد إلى التابع. عند وصول هذه الأرقام إلى العداد يتم إظهارها على دايوهات ضوئية موصولة على البوابة D. عند وصول أي بايت من السيد للتابع يقوم التابع بإرسال البايت 0x7E إلى المتحكم السيد دلالة على سلامه الاستقبال. عندما يستقبل المتحكم السيد البايت 0x7E يضيء دايد أحمر عند المتحكم السيد. التراسل، وفي حالة عدم وصول البايت 0x7E أو وصول بايت آخر نضيء دايد أخضر دلالة على سلامه. البرنامج التالي هو البرنامج الذى سيكتب عند المتحكم السيد (الماستر):

```
/*
 * SPI Master.c
 *
 * Created: 9/21/2017 8:09:39 PM
 * Author : Mohamed Eladawy
 */
#define F_CPU 1000000
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
```

```
#define ACK 0x7E
int main(void)
{
    DDRB = (1<<DDB5)|(1<<DDB3);      //Set MOSI, SCK as Output
    SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0); //Enable SPI, Set as Master
    //Prescaler: Fosc/16, Enable Interrupts
    DDRD |= 0xFF;                      //PORTD as Output
    unsigned char data;                //Received data stored here
    uint8_t x = 0;                     //Counter value which is sent

    while (1)
    {
        data = 0x00;                  //Reset ACK in "data"
        SPDR = ++x;                  //Load data into the buffer
        while(!(SPSR & (1<<SPIF))); //Wait until transmission complete
        data=SPDR;                   //coming data from slave
        if(data == ACK) {            //Check condition
            //If received data is the same as ACK, light green LED
            PORTD=0x01;
        }
        else {
            //If received data is not ACK, light Red LED
            PORTD=0x02;
        }
        _delay_ms(500);              //Wait
    }
}
/*
البرنامج التالي هو الذي سيكتب عند المتحكم التابع:
```

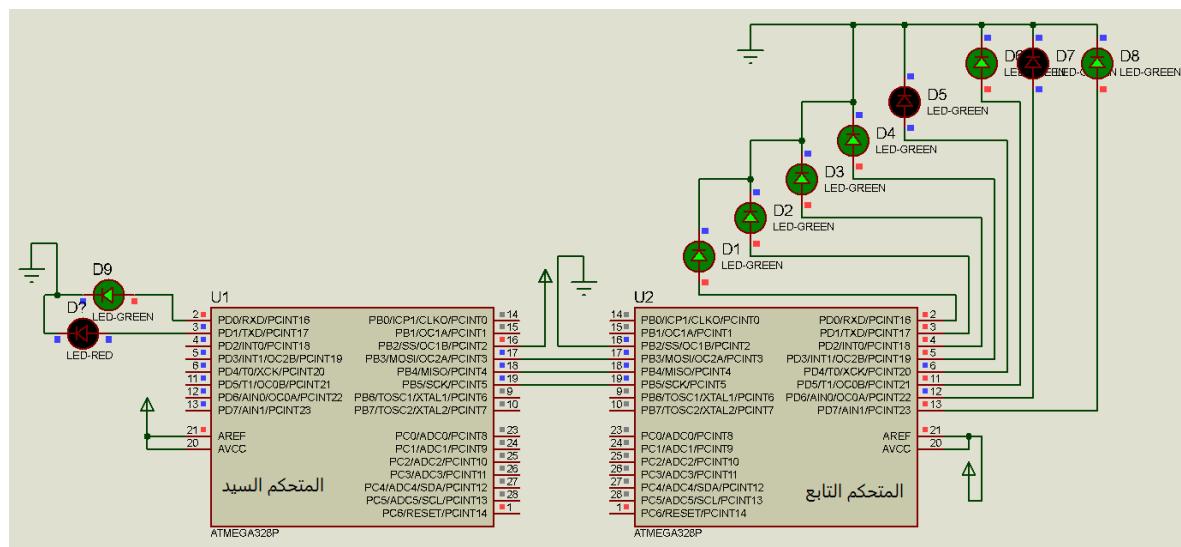
```
* SPI Slave.c
*
* Created: 9/21/2017 8:51:49 PM
* Author : Mohamed eladawy
*/
#define F_CPU 1000000
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#define ACK 0x7E

int main(void)
{
    DDRB=(1<<DDB4); //MISO as OUTPUT
    DDRD=0xFF;      //PORTD output
    SPCR=(1<<SPE); //Enable S
    while (1)
    {
        SPDR = ACK; //Load data into buffer
        while(!(SPSR & (1<<SPIF))); //Wait until transmission complete
        PORTD=SPDR; //Put received data on PORTD
        _delay_ms(500); //Wait
    }
}
```

إرسال ACK إلى المتحكم السيد بعد كل استقبال لأى بايت من السيد

تذكر أنه مع كل استقبال لأى بت من السيد عند التابع يتم إرسال بت من التابع إلى السيد بطريقة دائمة كما ذكرنا.

شكل ٧-١٢ يبين تنفيذ هذا المثال على البروتس.



شكل ٧-١٢ توصيل المتحكم السيد مع متحكمتابع والتواصل بينهما من خلال المسار SPI

## ملخص الفصل

لقد قدمنا في هذا الفصل طريقة التراسل للتراسل المتتابع، مع تنفيذ مثال على التراسل بين اثنين متتحكم يعمل أحدهما كسيد والأخر هو التابع. هذه الطريقة هي المستخدمة في التراسل بين المتتحكم ومعظم دوائر حرق البرنامج على الشريحة. لذلك فإن هذه الطريقة تستخدمن في التراسل مع الأجهزة المجاورة للمتحكم أو القريبة منه مساحيا.

## الفصل ١٣

### الاتصالات التتابعية غير المتزامنة

### Universal Synchronous Asynchronous Receiver/Transmitter, USART

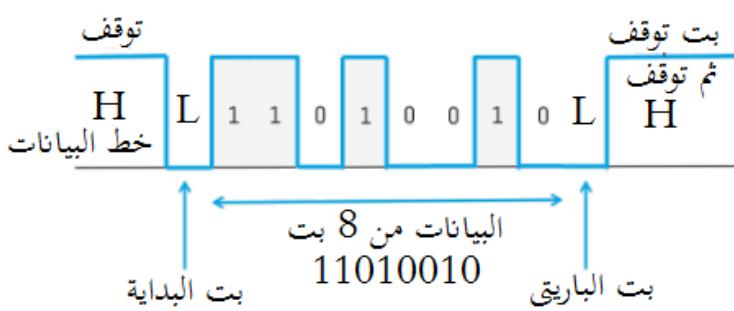
العناوين المضيئة في هذا الفصل:

- ١- مسجل بيانات التراسل
- ٢- مسجل الحالة والتحكم A للتراسل
- ٣- مسجل الحالة والتحكم B للتراسل
- ٤- مسجل الحالة والتحكم C للتراسل
- ٥- مسجل تحديد معدل التراسل (بود)
- ٦- تجهيز الشريحة USART للعمل
- ٧- اختبار تشغيل الشريحة USART

## ١-١٣ مقدمة

**تسمح** تقنية التراسل غير المتزامن بنقل البيانات بين المرسل والمستقبل دون الحاجة إلى وجود نبضات تزامن clock بين الطرفين لضمان التزامن في إرسال واستقبال البثات المتتالية. توجد شريحتان من الشرائح المنفصلة التي تقوم بهذا النوع من التراسل، الشريحة الأولى تسمى Universal Asynchronous Receiver/Transmitter، UART وتعنى الإرسال والاستقبال غير المتزامن العام، والشريحة الثانية تسمى Universal Synchronous Asynchronous Receiver/Transmitter، USART وتعنى الإرسال والاستقبال المتزامن وغير المتزامن العام. وهذه الشريحة يمكن شراؤها منفصلة وتوصيلها مع أي جهاز يحتاج لهذا النوع من التراسل كما هو الحال في الكثير من المعالجات التي لا تحتوى مثل هذه الشرائح بداخلها. طبعاً معظم المتحكمات AVR تحتوى هذه الشريحة مبنية بداخلها مثلها مثل المولات التماضية الرقمية وفي هذه الحالة فإن كل ما تحتاجه هو كيفية التعامل مع هذه الشريحة من خلال المتحكم عن طريق بعض الأوامر البسيطة لوضع وحيد وأصفار في مسجلات التحكم لهذه الشريحة. المتحكم atmega328 به شريحة من النوع USART ومهمتنا في هذا الفصل هي شرح كيفية التعامل مع هذا النوع من التراسل من خلال الشريحة USART الموجودة في المتحكم atmega328 مع التطبيق على ذلك باستخدامها في كل من نوعي التراسل غير المتزامن والمترادف.

لقد رأينا في الفصل ١٢ أنه في التراسل المتزامن يتم إخراج البيانات على التوالي على أحد الطرفين MOSI أو MISO على حسب كون الشريحة مخصصة لتكون هي السيد master أو التابع slave. البيانات في هذه الحالة تخرج متزامنة مع نبضات تزامن يكون مصدرها هو السيد. بمعنى أن كل بت من بثات البيانات تخرج متزامنة مع الحافة الصاعدة أو النازلة من نبضة التزامن، ومن هنا تكون سلامة البيانات مضمونة تماماً. هنا، وفي حالة التراسل غير المتزامن لا توجد نبضات



شكل ١-١٣ الشكل العام لإطار البيانات في حالة التراسل غير المتزامن

تزامن تربط ما بين المرسل والمستقبل، ولكن في حالة الإرسال transmission يتم وضع بثات البيانات بالتتابع على طرف يسمى طرف الإرسال Tx، ويتم استقبال البيانات بالتتابع على طرف آخر يسمى طرف الاستقبال receive أو Rx. هنا يبرز سؤال مهم، كيف يتم تحديد بداية ونهاية كل بait بين المرسل والمستقبل،

وهذا أمر مهم جدا لا تكون سلامة التراسل مضمونة بدونه. لحل هذه المشكلة فإنه في حالة التراسل غير المتزامن يتم إضافة بثات زائدة overheads في بداية ونهاية بait البيانات من خلالها يمكن تحديد بداية ونهاية البايت المرسلة. بait البيانات مع هذه البتات الإضافية تسمى الإطار frame، وعدد بثات الإطار يتم تحديده والاتفاق عليه بين المرسل والمستقبل، وهناك أكثر من شكل للإطار يمكن اختياره من قبل المستخدم. شكل ١-١٣ يبين رسميا تخطيطيا لبيانات البيانات والبتات الإضافية المضافة. لاحظ في هذا الشكل أنه عندما يكون خط البيانات في حالة توقف أو خامل idle فإنه يكون واحد أو على الجهد H, high. أول بت في الإطار من ناحية اليسار تسمى بت البداية start bit وهي أول نزول لخط البيانات للصفر بعد حالة التوقف. وهذا يعني أن البيانات ستبدأ فورا بعد هذه البت التي تساوى صفر دائما. يأتي بعد ذلك البتات الخاصة بالبيانات وهي عادة تكون بait كاملة (٨ بت أو أقل في بعض الأحيان)، وتأتي البت ذات القيمة الصغرى LSB أولا ثم الـ MSB. يأتي بعد ذلك بت أو اثنين أو أكثر خاصة بالباريتي parity، وهذه البتات تستخدم في تحديد إذا كان حدث خطأ في إرسال البيانات أم لا. تحديد هذا الخطأ موضوع كبير ولا مجال للكلام عنه بالتفصيل هنا ولكن سنكتفي بالكلام عنه باختصار فيما بعد. بعد بثات الباريتي تأتي بت أو اثنان عالية المستوى (١) تسمى بثات التوقف stop bits، وهذه تعني نهاية إرسال هذا الإطار من البيانات. بعد ذلك، إما أن يتم البدأ في إرسال إطار جديد بتزيل خط البيانات للصفر لإعطاء بت بداية start bit جديدة، أو أن يظل الخط عالي (١ أو idle) وهذا يعني توقف التراسل على الخط إلى أن يبدأ إطار جديد من البيانات.

هذا الذي سبق من تحديد الإطار بيت للبداية start bit وأخرى للنهاية stop bit ليس كافيا للتحديد الكامل للإطار حيث أن ذلك لم يحدد زمنا يحتوي هذا الإطار، إننا فقط حددنا شكل بداية ونهاية الإطار، ولكننا لم نحدد كم مقدار الزمن الذي يشغله هذا الإطار حتى نحدد من هذا الزمن بداية ونهاية كل بت (أو العرض الزمني لكل بت) داخل هذا الإطار. إن هذا يتحدد بما يسمى معدل بود Baud rate. معدل بود هو معدل إرسال البيانات ووحدته هي البت لكل ثانية (بت/الثانية أو bps). لذلك لابد من أن يعمل كل من المرسل والمستقبل عند نفس معدل البود، وإلا لن تتم عملية التراسل بكفاءة على الإطلاق، حيث سيتم فقد الكثير من البتات بينهما. اختيار قيمة معدل بود ليس عملية اختيارية مطلقة، ولكن يتم اختيار هذه القيمة من بين قيم معيارية معروفة وهي 2400 أو 4800 أو 9600 أو 19200 أو 38400 وهكذا.

لكى تبدأ عملية تراسل بين مرسل ومستقبل عليك اتباع الخطوات التالية التي سنراها بعد قليل بالتفصيل في معرض الحديث عن المسجلات المستخدمة في ذلك في المتحكم atmega328

- ١ - حدد معدل بود، ويجب أن يكون هو نفسه عند كل من المرسل والمستقبل ومن القيم القياسية المحددة مسبقاً، وسنجري تفاصيل لذلك بعد قليل.

PC6	1	28	PC5
(RXD) PD0	2	27	PC4
(TXD) PD1	3	26	PC3
PD2	4	25	PC2
PD3	5	24	PC1
(XCK) PD4	6	23	PC0
VCC	7	22	GND
GND	8	21	AREF
PB6	9	20	AVCC
PB7	10	19	PB5
PD5	11	18	PB4
PD6	12	17	PB3
PD7	13	16	PB2
PB0	14	15	PB1
Atmega328			

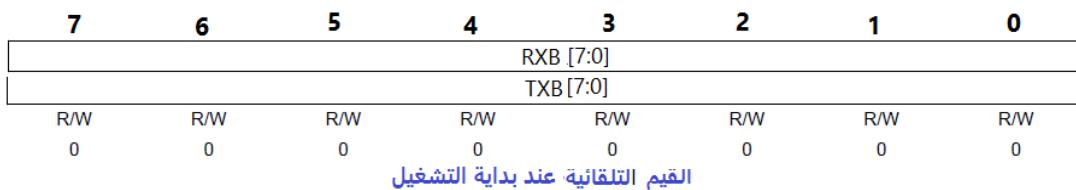
شكل ٢-١٣ الأطراف المستخدمة في التراسل غير المتزامن

- ٢ - حدد عدد برات البيانات في كل إطار.
- ٣ - في حالة المرسل، إنقل البيانات المراد إرسالها إلى مسجل البيانات أو العازل الذي سيتم إزاحة البيانات منه إلى الطرف الآخر. وفي حالة المستقبل تأكد من قراءة البيانات إلى وصلت للعازل حتى لا تكتب عليها البيانات الجديدة.
- ٤ - قم بتنشيط الشريحة لوضع المرسل أو المستقبل.

شكل ٢-١٣ يبين أطراف المتحكم atmega328 المستخدمة في هذه الحالة من التراسل، حيث نلاحظ وجود ثلاث أطراف فقط تستخدم في هذه العملية وهي الطرف TXD الذي تخرج عليه البيانات التابعة في حالة تخصيص المتحكم كمرسل، والطرف RXD الذي يتم استقبال البيانات عليه في حالة تخصيص المتحكم كمستقبل، وطرف التزامن XCK وهذا الطرف يستخدم فقط في حالة التراسل المتزامن باستخدام الشريحة USART.

بذلك يمكننا البدأ في دراسة المسجلات المستخدمة في عملية التراسل غير المتزامن بالتفصيل، ومن خلال ذلك سنفهم الكثير عن هذا النوع من التراسل.

## ٢-١٣ مسجل بيانات التراسل UDR



شكل ٣-١٣ مسجل بيانات التراسل في الشريحة USART

مسجل بيانات التراسل UDR هو في الحقيقة مسجلين لهما نفس العنوان ويتم التعامل معهما بنفس الإسم، UDR، ولكن في حالة الإرسال يتم وضع (كتابة) البيانات المراد إرسالها في المسجل TXB، وفي حالة الاستقبال يتم قراءة البيانات المرسلة من المسجل RXB. إذن هما مسجلين منفصلين بنفس الإسم ولكن أحدهما ينشط مع الكتابة TXB والآخر ينشط مع القراءة RXB.

سنرى بعد قليل أن مجال البيانات في الإطار من الممكن أن يكون ٥ أو ٦ أو ٧ ببات (وليس ٨ كالعادة)، حيث في هذه الحالة سيتم إهمال الباتات الأخرى غير المستخدمة عن طريق المرسل ووضعها بأصفار عند المستقبل. هناك علم إسمه UDRE Data Register Empty، عندما يكون واحد يعني أن المسجل UDR جاهز لاستقبال بيانات تمهدًا لإرسالها على الطرف TXB، وأى بيانات تكتب في المسجل UDR في حالة أن العلم UDRE يساوى صفر سيتم إهمالها ولن ترسل.

### ١٣-٣ مسجل الحالة والتحكم A للتراسل

مسجل الحالة والتحكم A في الشريحة USART واسمها UCSRA يتكون من ثمانى باتات كما في شكل ١٣-٤ وظيفتها كالتالى:

٧	٦	٥	٤	٣	٢	١	٠
RXC0	TXC0	UDRE0	FEO	DOR0	UPE0	U2X0	MPCM0
٠	٠	١	٠	٠	٠	٠	٠

القيم الثلائية عند بداية التشغيل

شكل ١٣-٤ مسجل الحالة والتحكم A للتراسل

#### البت ٧ RXC USART Receive is Complete

هذه البت أو العلم يكون واحد طالما أن هناك بيانات لم يتم قراءتها في مسجل بيانات الاستقبال. بمجرد قراءة البيانات من مسجل بيانات الاستقبال يصبح هذا العلم يساوى صفرًا. إذا تم إخماد المستقبل، فإن مسجل بيانات الاستقبال يتم مسحه ويصبح العلم RXC0 يساوى صفرًا. يمكن استخدام هذا العلم في بدء برنامج خدمة مقاطعة عند استكمال عملية استقبال بيانات كما سنرى بعد قليل.

**البت 6 TXC علم استكمال الارسال :USART Transmit is Complete**

يتم وضع هذا العلم بوحدة عند استكمال إزاحة إطار كامل من البيانات خارج مسجل بيانات التراسل UDR ولا يوجد أى بيانات جديدة في هذا المسجل. يمكن استخدام هذا العلم في بدء برنامج خدمة مقاطعة عند استكمال عملية إرسال بيانات كما سنرى بعد قليل.

**البت 5 UDRE مسجل بيانات التراسل فاضي :USART Data Register is Empty**

هذا العلم عندما يكون واحد يوضح أن مسجل بيانات التراسل فاضي ومستعد لاستقبال بيانات جديدة لإرسالها إلى على الطرف Tx . هذا العلم يكون واحد بعد إعادة وضع المتحكم reset دلالة على الاستعداد لاستقبال بيانات جديدة.

**البت 4 FE خطأ إطارى :Frame Error**

هذه البت تصبح واحد إذا كان هناك خطأ إطارى في البيانات الموجودة في مسجل بيانات الاستقبال، بمعنى أن بت التوقف stop bit تساوى صفر في الحرف الذى تم استقباله (تذكر أن بت التوقف تساوى واحد إذا كانت صحيحة). هذه البت FE تساوى صفر عندما تكون بت التوقف تساوى واحد.

**البت 3 DOR تجاوز البيانات :Data OverRun**

يحدث تجاوز للبيانات Data Over Run إذا كان مسجل البيانات مملوء (به حرف) وتم استقبال بداية حرف جديد بأن أصبحت بت البدأ Start bit تساوى صفر، في هذه الحالة تصبح البت DOR تساوى واحد. تظل هذه البت تساوى واحد حتى تتم قراءة مسجل بيانات التراسل UDR .

**البت 2 UPE خطأ باريتي :Parity Error**

هذه البت تساوى واحد إذا كان الحرف أو البايت الذى تم استقبالها تحتوى خطأ باريتي، وبالطبع عندما يكون اختبار الباريتي منشطا. هذه البت تظل محققة إلى أن تتم قراءة مسجل بيانات التراسل.

كما ذكرنا من قبل فإن بذات الباريٰت هي أبسط طريقة لاختبار أخطاء التراسل. الباريٰت هي عدد الوحاید في أي بآیت، فمثلاً البايت 01010101 بها أربع وحاید وبالتالي فإن هذه الباريٰت تكون زوجية، وأما البايت 01110110 فيها خمس وحاید وبالتالي فإن الباريٰت الخاصة بها تكون فردية.

الآن سنضيف بت للباريٰت على يمين أي بآیت من البايتات السابقة، متى نضع هذه البت تساوى واحد ومتي نضعها بصفر؟ هناك نظامان لذلك، نظام الباريٰت الزوجية، ونظام الباريٰت الفردية:

- نظام الباريٰت الزوجية:** في هذا النظام نضع بت الباريٰت بواحد أو صفر بحيث تجعل الباريٰت الكلية للبايت بعد إضافة بت الباريٰت زوجية. فمثلاً البايت 01010101 ستتصبح 010101010 بعد إضافة بت الباريٰت بصفر على يمين البايت الأصلية بحيث تظل الباريٰت للبايت الجديدة بعد إضافة بت الباريٰت زوجية (أربع وحاید). أما البايت 01110110 فتم إضافة بت الباريٰت فيها تساوى واحد لكن تكون الباريٰت الكلية للبايت بعد إضافة بت الباريٰت زوجية (ستة وحاید).
- نظام الباريٰت الفردية:** في هذا النظام نضع بت الباريٰت بواحد أو صفر بحيث تجعل الباريٰت الكلية للبايت بعد إضافة بت الباريٰت فردية. فمثلاً البايت 01010101 ستتصبح 010101011 بعد إضافة بت الباريٰت بواحد على يمين البايت الأصلية بحيث تصبح الباريٰت للبايت الجديدة بعد إضافة بت الباريٰت فردية (خمس وحاید). أما البايت 01110110 فتم إضافة بت الباريٰت فيها صفر لكن تظل الباريٰت الكلية للبايت بعد إضافة بت الباريٰت فردية (خمسة وحاید).

الجدول ١-١٣ يبيّن كيفية إضافة بت الباريٰت في النظامين الزوجي والفرد لبعض الأرقام الإضافية كأمثلة:

جدول ١-١٣ أمثلة على كيفية إضافة بت الباريٰت في حالة النظام الزوجي والفرد للباريٰت

البايت الأصلية	عدد الوحاید	في حالة النظام الزوجي	في حالة النظام الفردی
00000000	0	000000000	000000001
11011101	6	110111010	110111011
10110110	5	101101101	101101100
11111111	8	111111110	111111111

السؤال الآن هو: لماذا كل ذلك؟ الإجابة باختصار أن ذلك يمكننا من طريقة بسيطة لاكتشاف أخطاء التراسل. إفترض مثلاً أن لدينا البايت 11011101 (ستة وحاید) ونريد إرسالها في نظام الباريٰت الزوجي. في هذه الحالة ستتصبح البايت

بعد إضافة بت الباريتي هي 110111010 كما في الجدول ١-١٣ ، وسيتم هذا الإطار إلى الطرف الآخر بعد إضافة بت البداية وبت التوقف. المفروض أن المستقبل يعرف أن النظام الزوجي للباريتي هو المستخدم، ولذلك فإن المستقبل يقوم بعد الوحيد في الإطار بعد استقباله بما في ذلك بت الباريت. فإذا وجد عدد الوحيد زوجي فإن ذلك يعني أن البایت التي تم استقبالها غالباً صحيحة. ولكن بفرض أنه أثناء التراسل تغيرت أحد البتات من صفر إلى واحد (أو العكس) كما في البایت التي باللون الأحمر في البایت السابقة 1111010. في هذه الحالة أصبح عدد الوحيد فردي (سبعة) وحيث نظام الباريتي المعول به زوجي فهنا يحدث خطأ الباريتي ويصبح علم خطأ الباريتي يساوى واحد. بالطبع فإن هذه الطريقة تعتبر بسيطة كما ذكرنا لأنها لا تكتشف إلا خطأ واحد في البيانات ولكن إذا حدث أكثر من خطأ فهنا تكون طرقاً أكثر تطوراً في الكشف عن مثل هذه الأخطاء لأنها خارج موضوع الكتاب ويخصص له كتب كاملة. الخلاصة من ذلك هي أنه لابد من تحديد عدد بتات الباريتي المستخدمة ونظام الباريتي المعول به بين الطرفين وهذا ما سنراه بعد قليل.

### **البیت ١ U2X مضاعفة سرعة التراسل**

بوضع هذه البیت تساوى واحد يتم تخفيض معامل القسمة في المقام عند حساب معدل بود من ١٦ إلى ٨ (سترى ذلك بعد قليل عند ضبط قيمة معدل بود) وبذلك تتضاعف سرعة التراسل. لاحظ أن ذلك يحدث فقط في حالة التراسل غير المترادف.

### **البیت ٢ MPCM طريقة التراسل متعدد المعالجات**

بوضع هذه البیت تساوى واحد يتم تفعيل طريقة التراسل متعدد المعالجات. في هذه الطريقة يمكن توصيل أكثر من متحكم كتابع لمتحكم سيد وكلهم في مسار متتالي ويتم اختيار المتحكم المراد التعامل معه من خلال إطار عنوان يحمل عنوان للمتحكم المراد التعامل معه. نحن لن نفترض هذه الطريقة للتعامل لذلك سنقوم دائماً بوضع هذه البیت تساوى صفر تجنباً للتعامل مع هذه الطريقة من التراسل.

## **٤-١٣ مسجل الحالة والتحكم B للتراسل**

مسجل الحالة والتحكم B في الشريحة USART يتكون من ثمانى بتات كما في شكل ٥-١٣ وظيفتها كالتالى:

7	6	5	4	3	2	1	0
RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ02	RXB8	TXB8
0	0	1	0	0	0	0	0

القيم التلقائية عند بداية التشغيل

شكل ٥-١٣ مسجل الحالة والتحكم B للتراسل

#### البت 7 RXCIE تنشيط مقاطعة استكمال الاستقبال

بوضع واحد في هذه البت تصبح مقاطعة استكمال الاستقبال فعالة بحيث إذا أصبح علم استكمال الاستقبال RXC (البت ٧ في المسجل UCSRA) يساوى واحد، وكان علم تنشيط المقاطعة العام I في مسجل الحالة يساوى واحد أيضا، فإنه سيتم الانتقال إلى برنامج خدمة مقاطعة ISR لخدمة هذه المقاطعة.

#### البت 6 TXCIE تنشيط مقاطعة استكمال الإرسال

بوضع واحد في هذه البت تصبح مقاطعة استكمال الإرسال فعالة بحيث إذا أصبح علم استكمال الإرسال TXC (البت ٦ في المسجل UCSRA) يساوى واحد، وكان علم تنشيط المقاطعة العام I في مسجل الحالة يساوى واحد أيضا، فإنه سيتم الانتقال إلى برنامج خدمة مقاطعة ISR لخدمة هذه المقاطعة.

#### البت 5 UDRIE تنشيط مقاطعة مسجل البيانات الفارغ

##### :Interrupt Enable

بوضع واحد في هذه البت تصبح مقاطعة مسجل البيانات فارغ فعالة بحيث إذا أصبح علم مسجل بيانات التراسل فارغ UDRE (البت ٥ في المسجل UCSRA) يساوى واحد، وكان علم تنشيط المقاطعة العام I في مسجل الحالة يساوى واحد أيضا، فإنه سيتم الانتقال إلى برنامج خدمة مقاطعة ISR لخدمة هذه المقاطعة.

#### البت 4 RXEN Receiver Enable

وضع واحد في هذه البت يفعل مستقبل الشرحية USART. وضع صفر فيها سيحمد المستقبل وبالتالي فإن مسجل بيانات الاستقبال (عازل الاستقبال) سيتم إفراجه.

**البت 3 TXEN Transmitter Enable :**

وضع واحد في هذه البت يفعل مرسل الشريحة USART. وضع صفر فيها سيخمد المرسل ولكن هذا الإخماد لن يكون فعالاً إلا بعد استكمال إرسال أي بait حالية أو معلقة، أي عندما يكون مسجل بيانات الإرسال (أو عازل الإرسال) لا يحتوى أي بيانات يراد إرسالها.

**البت 2 UCSZ2 Character size :**

جدول ٢-١٣ تحديد عدد ببات البيانات (حجم الحرف) في كل إطار

UCSZ2 في المسجل UCSRB	UCSZ1 في المسجل UCSRC	UCSZ0 في المسجل UCSRC	حجم الحرف
0	0	0	٥ بت
0	0	1	٦ بت
0	1	0	٧ بت
0	1	1	٨ بت
1	0	0	محظوظ
1	0	1	محظوظ
1	1	0	محظوظ
1	1	1	٩ بت

هذه البت مع البت رقم 0 والبت رقم 1 في مسجل الحالة والتحكم C، UCSRC، والذي سيتم شرحه في الجزء التالي، سيحددان عدد الباتات الخاصة بالبيانات (حجم الحرف) في الإطار الذي سيستخدمه كل من المرسل والمستقبل تبعاً للجدول ٢-١٣.

**البت 1 RXB8 Receive Data Bit 8 (ال Batese) في البيانات المستقبلة :**

عند استخدام إطار بيانات، عند المستقبل، به ٩ باتات كما في الجدول ٢ فإن البت التاسعة تكون هي هذه البت RXB8. هذه البت يتم قراءتها أولاً ثم يتم قراءة باقي الباتات الأخرى من مسجل بيانات المستقبل UDR.

**البت ٨ TXB8 Transmit Data Bit 8 (ال Batese) في البيانات المرسلة:**

عند استخدام إطار بيانات، عند المرسل، به ٩ برات كما في الجدول ٦-١٣ فإن البت التاسعة تكون هي هذه البت TXB8. هذه البت يتم كتابتها أولاً ثم يتم كتابة باقي البوت الأخرى في مسجل بيانات المرسل UDR.

**٦-٥ مسجل الحالة والتحكم C للتراسل**

مسجل الحالة والتحكم C في الشريحة USART واسمها UCSRC يتكون من ثمان برات كما في شكل ٦-١٣ وظيفتها كالتالي:

٧	٦	٥	٤	٣	٢	١	٠
UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01/ UDORD0	UCSZ00/ UCPHAO	UCPOLO
٠	٠	١	٠	٠	٠	٠	٠

القيم التلقائية عند بداية التشغيل

شكل ٦-١٣ مسجل الحالة والتحكم C للتراسل

**البت ٦ UMSEL01 USART Mode USART** **البت ١ لاختيار طريقة عمل الشريحة Select**

هذه البت مع البت ٦ في نفس هذا المسجل يتم بحث اختيار طريقة عمل الشريحة USART كما في الجدول ٦-٣.

جدول ٦-١٣ برات اختيار طريقة عمل الشريحة USART

UMSEL01	UMSEL00	طريقة العمل
0	0	وضع الشريحة USART لتعمل بالطريقة غير المتزامنة
0	1	وضع الشريحة USART لتعمل بالطريقة المتزامنة
1	0	محجوز
1	1	سيد (ماستر) في حالة التراسل (MSPIM) SPI

## البت 6 UMSEL00 USART لاختيار طريقة عمل الشريحة USART :Mode Select

هذه البت مع البت 7 في نفس هذا المسجل يتم بحث اختيار طريقة عمل الشريحة USART كما في الجدول ١٣-٣.

## البنا ٥ و ٤ UPM1:0 Parity ووضع نوع وحالة الباريتي في الشريحة USART :Mode

هذه البنا تنشط وتضع نوع وحالة الباريتي التي سيعمل بها المتحكم. عند تنشيط الباريتي سيقوم المرسل بتوليد بنا الباريتي في الإطار المرسل مع بنا البيانات. بناء على ذلك يقوم المستقبل بتوليد قيمة الباريتي من البيانات المرسلة، فإذا توافقت مع بنا ال UPM المرسلة فهذا معناه عدم وجود خطأ، وإذا لم توافق فهذا معناه وجود خطأ وبناء عليه يتم وضع العلم UPE يساوى واحد في المسجل UCSRA. الجدول ١٣-٤ يبين نوع الباريتي بناء على وضع البنا UPM1:0.

جدول ١٣-٤ نوع وحالة الباريتي بناء على وضع البنا UPM1:0

UPM1	UPM0	نوع وحالة الباريتي
0	0	الباريتي غير منشطة (لا توجد بنا باريتي)
0	1	محجوز
1	0	الباريتي مفعلة، باريتي زوجية
1	1	الباريتي مفعلة، باريتي فردية

## البت 3 USBS اختيار عدد بنا التوقف في الشريحة USART :USBS Stop Bit Select

هذه البت يتم بحث اختيار عدد بنا التوقف التي يتم وضعها في نهاية الإطار عند المرسل كما في الجدول ١٣-٥. لاحظ أن بنا التوقف تكون إما بنا واحدة أو اثنين.

## البنا ٢ و ١ UCSZ1:0 Character USART اختيارات عدد حجم الحرف في الشريحة :Size

هذه البنا مع البت رقم ٢ UCSZ2 في المسجل UCSRB تحدد عدد البنا المستخدمة كبيانات في كل إطار كما في الجدول ٦-١٣ ولقد سبق شرح ذلك، وبالطبع فإن كل من المرسل والمستقبل يستخدمان هذه البنا.

جدول ٦-٥ اختيارات عدد بنا التوقف في الشريحة USART

USBS	عدد بنا التوقف
٠	بت واحدة
١	اثنان بت

## البنا ٠ UCPOL Clock Polarity USART قطبية نبضات التزامن في الشريحة

تستخدم هذه البت في حالة استخدام الشريحة USART في طريقة التراسل المتزامن. في حالة استخدام الشريحة في التراسل غير المتزامن يتم وضع هذه البت بصفر. هذه البت يتم بناء على قيمتها إذا كانت واحد أو صفر تحديد الحافة الفعالة لنبضات التزامن XCK كما في الجدول ٦-١٣ .

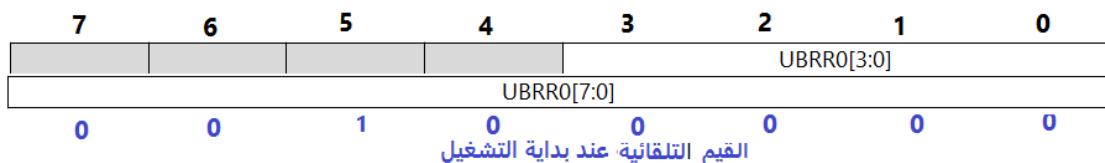
جدول ٦-١٣ تحديد الحافة الفعالة لنبضات التزامن في حالة التراسل المتزامن

UCPOL	البيانات المرسلة على الطرف TX تنوافق مع:	يتم قراءة البيانات المستقبلة على الطرف عند: Rx
٠	الحافة الصاعدة لنبضات التزامن XCK	الحافة النازلة لنبضات التزامن XCK
١	الحافة النازلة لنبضات التزامن XCK	الحافة الصاعدة لنبضات التزامن XCK

## ٦-١٣ مسجل تحديد معدل بود (معدل التراسل) في الشريحة USART

هذا المسجل يتم استخدامه لتوليد معدل بود الذي سيعمل عنده كل من المرسل والمستقبل باليت/الثانية. هذا المسجل يتكون من ١٦ بت منها آخر ٤ بنا غير مستخدمة، وبالطبع طالما أنه ١٦ بت فإنه سيمثل في مسجلين كل منهما ٨ بت كما في شكل ٧-١٣ . هذان المسجلان هما UBRRH وهو مستخدم بالكامل، و UBRRRL ويستخدم

منه فقط أول ٤ بت والأربعة الأخرى غير مستخدمة. إذن هناك ١٢ بت من المسجل UBRR هي التي يتم استخدامها في الحصول على معدل بود، والقيمة العددية لهذه البتات يتم التعويض بها في الجدول ٧-١٣ للحصول على قيمة البو德 الحقيقية بمعرفة تردد نبضات التزامن الخاصة بالمحكم fosc. هناك عداد ثانئ من ١٢ بت يتم تحديده بقيمة المسجل UBRR حيث يبدأ هذا العداد في التناقص بمقدار واحد إلى أن يصل صفرًا، حيث عندها تنتهي نبضة من نبضات البود، وبعدها يتم تحميل العداد مرة ثانية بمحطويات المسجل UBRR، وهكذا. إذن نبضات البود الناتجة من خرج هذا العداد يمكن كتابتها كما يلى:  $(UBRR+1)fosc$ . هذه النبضات الناتجة يتم قسمتها على ٢ لتصبح هي نبضات التزامن التي يتم استخدامها في حالة استخدام الشريحة USART في الحالة المتزامنة. يتم أيضًا قسمتها على ١٦ لتكون هي معدل بود العادي مع وضع بت مضاعفة السرعة U2X تساوى صفرًا، أو قسمتها على ٨ لتعطى معدل بود مضاعف أو سرعة تراسل مضاعفة عند وضع U2X تساوى واحد، وكل ذلك بالطبع في حالة التراسل غير المتزامن.



شكل ٧-١٣ مسجل تحديد معدل بود في الشريحة USART

جدول ٧-١٣ حساب معدل بود عند الحالات المختلفة لتشغيل الشريحة USART

حالة تشغيل الشريحة USART	حساب معدل بود بدلاً من UBRR	حساب UBRR بدلاً من معدل بود
الوضع العادي غير المتزامن (U2X=0)	$BAUD = \frac{fosc}{16(UBRR + 1)}$	$UBRR = \frac{fosc}{16BAUD} - 1$
وضع السرعة مضاعفة غير المتزامن (U2X=1)	$BAUD = \frac{fosc}{8(UBRR + 1)}$	$UBRR = \frac{fosc}{8BAUD} - 1$
الوضع المتزامن حيث ستكون الشريحة هي الماستر أو السيد	$BAUD = \frac{fosc}{2(UBRR + 1)}$	$UBRR = \frac{fosc}{2BAUD} - 1$

\*في هذا الجدول: BAUD هو معدل بود بالبت في الثانية، و UBRR هي محطويات مسجل تحديد معدل بود (١٢ بت) قيمته من صفر حتى ٤٠٩٥ (١٢-١)، و fosc هي تردد نبضات تزامن المحكم.

## ٧-١٣ تجهيز الشريحة USART للعمل في المتحكم atmega328

قبل البدأ في استخدام الشريحة USART لابد من تجهيزها أولاً تبعاً للخطوات التالية:

- ١ - تحديد معدل بود
- ٢ - تحديد حجم البيانات في الإطار (٥ أو ٦ أو ٧ أو ٨ أو ٩ بتات)
- ٣ - تنشيط الاستقبال والإرسال أو أي منهما
- ٤ - تحديد الباريتي وعدد بتات التوقف

الأوامر التالية ستقوم بهذه المهمة:

```
#define BAUD 9600
```

هذا الأمر يحدد معدل بود، وكما قلنا أننا نختار من بعض القيم القياسية مثل 9600 و 19200 و 38400 بت في الثانية، وهكذا

```
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)
```

حساب القيمة التي ستوضع في مسجل تحديد معدل بود UBRR. F\_CPU هي تردد نبضات تراثن المتحكم fosc في الجدول (٧-١٣).

```
UBRR0H = (BAUDRATE>>8);
```

إزاحة محتويات القيمة BAUDRATE ثمان بتات ناحية اليمين بحيث يتم وضع البتات الزائدة عن الشمانية فقط في الجزء الأعلى من المسجل UBRRL و UBRRH.

```
UBRR0L = BAUDRATE;
```

وضع الشمان بتات الأولى من القيمة BAUDRATE في المسجل UBRRL.

```
UCSR0B |= (1<<TXEN0)|(1<<RXEN0);
```

تنشيط الإرسال والاستقبال بوضع واحد في البتات TXEN و RXEN في المسجل UCSRB

```
UCSR0C |= (1<<UCSZ00)|(1<<UCSZ01);
```

تحديد بتات البيانات بثمانية بتات من خلال المسجل UCSRC

عند إرسال بيانات من شريحة تعمل كمرسل يمكن استخدام الأوامر التالية:

```
while (!(UCSR0A & (1<<UDRE0)));
```

انتظار حتى يفرغ مسجل بيانات الإرسال UDR.

```
UDR0 = data;
```

تحميل البيانات data في مسجل بيانات الإرسال UDR

عند استقبال بيانات من شريحة تعمل كمستقبل يمكن استخدام الأوامر التالية:

```
while(!(UCSR0A) & (1<<RXC0));
```

انتظار حتى يتم استكمال استقبال كل البيانات

```
Data=UDR0;
```

وضع البيانات التي تم استقبالها في المسجل UDR في المتغير data

## ٨-١٣ اختبار تشغيل الشريحة USART

بعد أن رأينا كيفية تجهيز الشريحة للعمل في الجزء السابق يمكننا الآن نجرب عليها أكثر من مثال لاختبارها.

### ١-تشغيل الشريحة USART للإرسال والاستقبال مع نفسها

سنقوم هنا بتوصيل طرف الإرسال TX على طرف الاستقبال RX بحيث أن أي بيانات يتم إرسالها على الطرف TX يتم استقبالها على الطرف RX. في هذا المثال سنبدأ ببيانات الإرسال بالقيمة 0x00 وبعد استقبال هذه البيانات على الطرف RX نجمع واحد عليها ونعرضها على البوابة B. أى أن البوابة B في هذه الحالة ستكون بمثابة عدد ثانى يمكننا متابعته على الخرج كما في شكل ٨-١٣. البرنامج الذى سينفذ ذلك سيكون كما يلى:

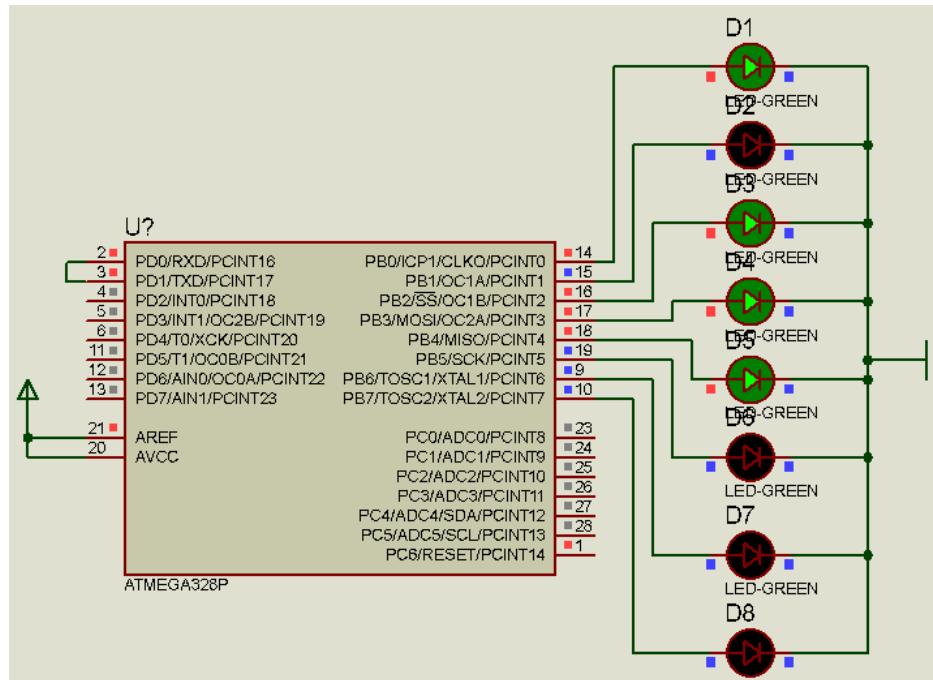
```
/*
* USART0TxRx.c
*
* Created: 10/5/2017 7:29:56 AM
* Author : Mohamed Eladawy
*/
```

```
#define F_CPU 1000000
#include <avr/io.h>
#define BAUD 9600
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)
#include <util/delay.h>
```

تحديد معدل بود بالقيمة ٩٦٠٠ وحساب محتويات المسجل .UBRR0

```
int main(void)
```

```
{
```



شكل ٨-١٣ الإرسال والاستقبال في نفس الشريحة USART

```
unsigned char data;
```

```
data=0x00;
```

```
DDRB=0xFF;
```

```
UBRR0H = (BAUDRATE>>
```

تحديد البايت العليا والصغرى من مسجل الود UBRR0K، ثم تفعيل الإرسال والاستقبال، ثم اختيار عدد بات البيانات يساوى ثمانية، فيما عدا ذلك لن يكون هناك باريتي سيكون هناك بت توقف واحدة (قيمة تلقائية).

```
UBRR0L = BAUDRATE;
```

```
UCSR0B |= (1<<TXEN0)|(1<<RXEN0);
```

```
UCSR0C |= (1<<UCSZ00)|(1<<UCSZ01);
```

```
while (1)
```

```
{
```

```
    while (!(UCSR0A & (1<<UDR0)),
```

```
        UDR0 = data;
```

انتظار أن يكون مسجل إرسال البيانات فارغ ثم إرسال البيانات.  
ثم انتظار استكمال استقبال البيانات.

```

while(!(UCSR0A & (1<<RXC0)));
data=UDR0;
PORTB=data;
_delay_ms(500);
data++;
}
}

```

قراءة البيانات المستقبلة، ثم إخراجها على البوابة B، ثم التأخير نصف ثانية، ثم زيادة البيانات بواحد والتكرار.

## ٢- الإرسال والاستقبال بين متحكمين عبر الشريحة USART

في هذا المثال سنقوم بربط شريحتين USART عن طريق توصيل طرف الإرسال Tx في الشريحة الأولى بطرف الاستقبال Rx في الشريحة الثانية، وطرف الاستقبال Rx في الشريحة الأولى بطرف الإرسال Tx في الشريحة الثانية كما في شكل ٩-١٣. سنكتب برنامج عند كل من المرسل والمستقبل بقرأ البيانات المستقبلة وعرضها على البوابة الخاصة به، ثم يزيد عليها واحد ويعيدها إرسالها للطرف الآخر. لذلك سنجد أن أحد المعالجين سعيد بدءاً من الصفر ويعرض القيمة الزوجية فقط، بينما المتحكم الآخر سيبدأ من الواحد ويعرض القيم الفردية فقط. البرنامج عند كل من المرسل والمستقبل سيكونان كما يلى:

البرنامج عند المرسل سيكون كما يلى مع ملاحظة أنه هو نفسه البرنامج السابق تقريباً:

```

/*
* USART1Tx.c
*
* Created: 10/5/2017 7:29:56 AM
* Author : Mohamed Eladawy
*/
#define F_CPU 1000000
#include <avr/io.h>
#define BAUD 9600
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)
#include <util/delay.h>

```

```

int main(void)
{
    unsigned char data;
    data=0x00;
    DDRB=0xFF;
    UBRR0H = (BAUDRATE>>8);
    UBRR0L = BAUDRATE;
    UCSR0B |= (1<<TXEN0)|(1<<RXEN0);
    UCSR0C |= (1<<UCSZ00)|(1<<UCSZ01);

    while (1)
    {
        while (!(UCSR0A & (1<<UDRE0)));
        UDR0 = data;
        while (!(UCSR0A & (1<<RXC0)));
        data=UDR0;
        PORTB=data;
        _delay_ms(500);
        data++;
    }
}

```

والبرنامج عند المستقبل سيكون كما يلى والأمر لا يحتاج لتعليقات أخرى عن البرنامج الأول:

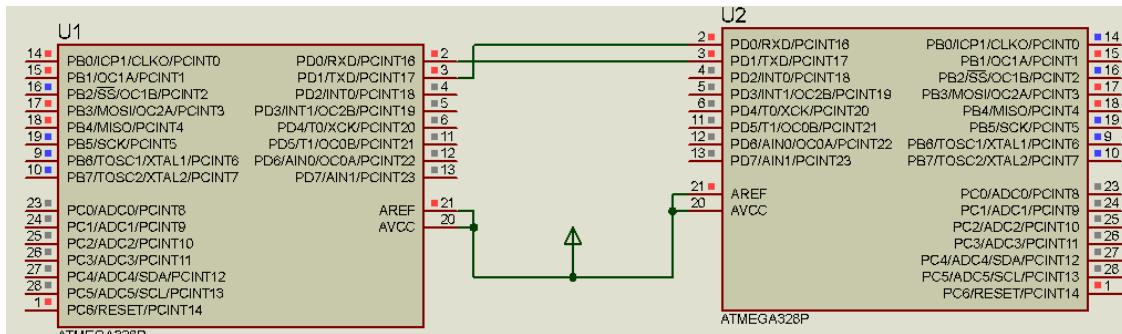
```

/*
* USART1Rx.c
*
* Created: 10/5/2017 8:22:23 AM
* Author :Mohamed Eladawy
*/
#define F_CPU 1000000
#include <avr/io.h>

```

```
#define BAUD 9600
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)
#include <util/delay.h>

int main(void)
{
    unsigned char data;
    data=0x55;
    DDRB=0xFF;
    UBRR0H = (BAUDRATE>>8);
    UBRR0L = BAUDRATE;
    UCSR0B |= (1<<TXEN0)|(1<<RXEN0);
    UCSR0C |= (1<<UCSZ00)|(1<<UCSZ01);
    while (1)
    {
        while(!(UCSR0A & (1<<RXC0)));
        data=UDR0;
        PORTB=data;
        _delay_ms(500);
        data++;
        while (!(UCSR0A & (1<<UDRE0)));
        UDR0 = data;
    }
}
```



شكل ٩-١٣ التراسل بين متحكمين من خلال الشريحة USART

في شكل ٩-١٣ لم نضع دايودات ضوئية على البوابة B في كل من المتحكمين وأكفينا بالمربيعات الحمراء التي يضعها البروتوس على أطراف البوابة B والتي تبين حالة كل طرف. لاحظ أن الخرج على المتحكم الأيمن هو ٢٦ (٠٠٠١١٠١٠) وعلى المتحكم الأيسر هو ٢٧ (٠٠٠١١٠١١)، مما يدل على سلامية التراسل.

### ٣- الإرسال والاستقبال بين متحكم والمخرج التتابعى للحاسوب

يمكن محاكاة المخرج التتابعى فى الحاسوب serial port فى برنامج البروتوس عن طريق ما يسمى بالطرف الافتراضى virtual terminal وهى أحد مكونات أيقونة الأجهزة الافتراضية virtual instruments التى تحتوى الأوسولوسكوب، والفولتميتر، وغير ذلك من الأجهزة الافتراضية. لذلك فإننا فى شاشة العمل الخاصة بالبروتوس سنضع الطرف الافتراضى والمتحكم atmega328 كما فى شكل ١٠-١٣، ثم سنقوم بتوصيل الطرف Rx فى الطرف الافتراضى بالطرف Tx فى المتحكم، والطرف Tx فى الطرف الافتراضى بالطرف Rx فى المتحكم كما فى الشكل. بعد ذلك سنكتب البرنامج التالى الذى سيقوم بعملية التراسل. عند تشغيل برنامج البروتوس يتم فتح شاشة صغيرة، أنقر على هذه الشاشة وابداً فى الكتابة. أى حرف تكتبه على هذه الشاشة سيخرج على الطرف Tx للمخرج المتالى، حيث يستقبله المتحكم على الطرف Rx ويقرأه. البرنامج الموجود فى متحكم يقرأ الحرف المرسل، ويجمع عليه واحد، ويضعه بين قوسين مربعين، ويرسله مرة ثانية إلى المخرج المتالى (الطرف الافتراضى). أى أننا إذا كتبنا على الشاشة الافتراضية الرقم ١، فإن المتحكم سيرد بالرقم [2]، وإذا كتبنا الرقم ٣، فإن المتحكم سيرد بالرقم [4]، وإذا كتبنا الحرف a، سيرد المتحكم بالرمز [b] وهكذا.

/\*

\* USARTCOMPORT.c

\*

\* Created: 10/5/2017 8:22:23 AM

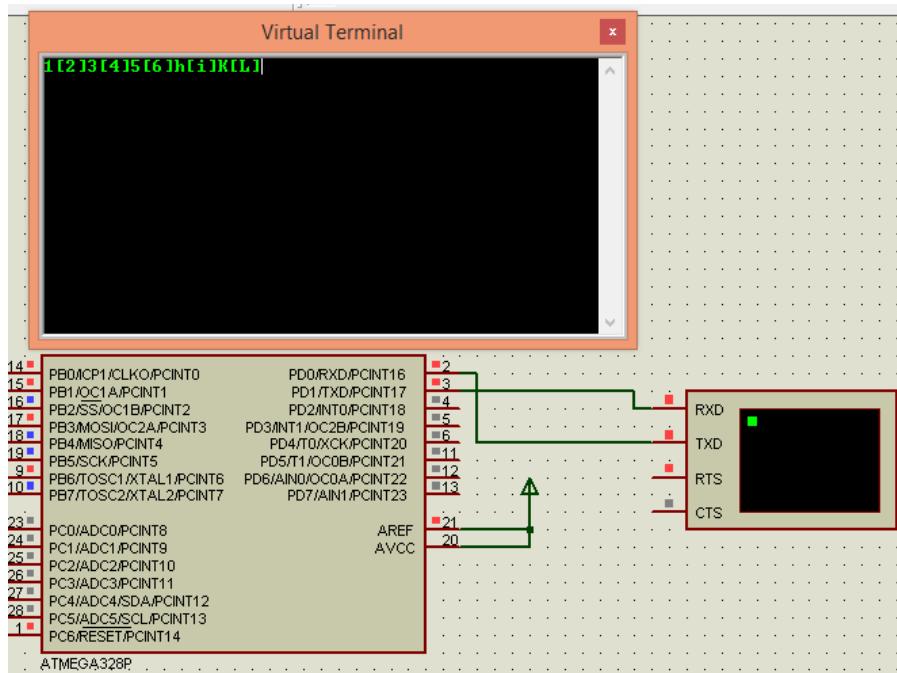
\* Author :Mohamed Eladawy

\*/

```
#define F_CPU 1000000
#include <avr/io.h>
#define BAUD 4800
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)
#include <util/delay.h>
int main(void)
{
    char data;
    DDRB=0xFF;
    UBRR0L = BAUDRATE;
    UBRR0H = (BAUDRATE>>8);
    UCSR0B |= (1<<TXEN0)|(1<<RXEN0);
    UCSR0C |= (1<<UCSZ00)|(1<<UCSZ01);
    while (1)
    {
        while(!(UCSR0A & (1<<RXC0)));
        data=UDR0;
        PORTB=data;
        _delay_ms(500);
        data++;
        while (!(UCSR0A & (1<<UDRE0)));
        UDR0 = '[';
        while (!(UCSR0A & (1<<UDRE0)));
        UDR0 = data;
        while (!(UCSR0A & (1<<UDRE0)));
        UDR0 = ']';
    }
}
```

}

}



شكل ١٠-١٣ توصيل المتحكم على طرف افتراضي يحاكي المخرج التتابعى

في البرامج السابقة كنا نختار قيمة معدل بود (Baud rate) من أحد القيم القياسية ولتكن مثلاً 9600، ثم نعرض بها في المعادلة  $F_{CPU}/16*9600$  ( $F_{CPU}$ ) لنحصل على القيمة التي يجب وضعها في المسجل BRR0 للحصول على هذا الבוד. القيمة الناتجة من هذه المعادلة يتمأخذ قيمتها الصحيحة فقط وإهمال الكسر الناتج منها. فمثلاً بالتعويض عن  $F_{CPU}=1000000$  ( $F_{CPU}$  واحد ميجا هرتز) سنحصل على القيمة 5.5104166. على ذلك سيتم وضع القيمة 5 فقط في المسجل BRR0 وإهمال الكسر المصاحب لها. هذا الكسر يعادل 10.4% وهي نسبة خطأ كبيرة. نتيجة لهذا الخطأ الكبير من الممكن أن يحدث خطأ في التراسل. لذلك فقد وجدنا أنه بوضع معدل بود يساوى 9600 في الطرف الافتراضي وفي برنامج المتحكم وجدنا أن التراسل لم يكن صحيحاً. لذلك وجب أن نختار معدل بود قياسي لا يعطى هذا الخطأ الكبير فكان اختيار المعدل 4800 في البرنامج السابق الذي أعطى 0.2% خطأ تقريباً وكان التراسل على ما يرام كما في شكل ١٠-١٣ .

## تمرين

يمكنك تطوير البرنامج السابق بحيث أن أي حروف يتم كتابتها على المخرج المتتالي (الطرف الافتراضي) يقوم المتحكم بعرضها على شاشة بللورة سائلة LCD يتم توصيلها على البوابة B.

## ملخص الفصل

لقد تم التعامل في هذا الفصل مع التراسل غير المتزامن من خلال استخدام الشريحة USART وهي أحد الملحقات المهمة في المتحكم atmega328 في هذا النوع من التراسل. تعرفنا في هذا الفصل على الفرق بين التراسل المتزامن وغير المتزامن، ثم دراسة مسجلات هذه الشريحة بالتفصيل واستخدامها في أكثر من مثال من أمثلة التراسل.

## الفصل ١٤

### فيوزات المتحكم atmega328

### Fuses of The Atmega328 Microcontroller

العناوين المضيئة في هذا الفصل:

- ١- مصادر نبضات تزامن المتحكم
- ٢- مسجل معامل تقسيم نبضات التزامن
- ٣- بيانات الفيوزات

## ١٤ - مقدمة

**لقد رأينا** في الفصول السابقة كيفية كتابة برنامج المتحكم من خلال المنصة أتمل استديو، وبعد الانتهاء من الكتابة والتأكد من أن البرنامج يعمل بصورة صحيحة وليس به أخطاء من خلال قائمة البناء Build كما رأينا في الكثير من الأمثلة في الفصول السابقة. الخطوة الثانية كانت هي بناء الدائرة في برنامج المحاكاة الإلكتروني بروتس من خلال وضع المتحكم وتوصيله بكل الدوائر الخارجية من موافير أو مظهرات أو غير ذلك من الحساسات. بعد ذلك كنا نقوم بنقل نسخة البرنامج المكتوبة بلغة الآلة (أو ملف المكسا hexadecimal file) من برنامج الأتمل استديو إلى شريحة المتحكم في البروتس. في النهاية كنا نقوم بتنفيذ البرنامج ونلاحظ نتيجة عمله، وإذا احتاج الأمر لبعض التعديلات في البرنامج كنا نقوم بتعديلها في الأتمل استديو والعودة مرة ثانية للبروتس، وهكذا حتى تعمل الدائرة بالصورة المطلوبة.

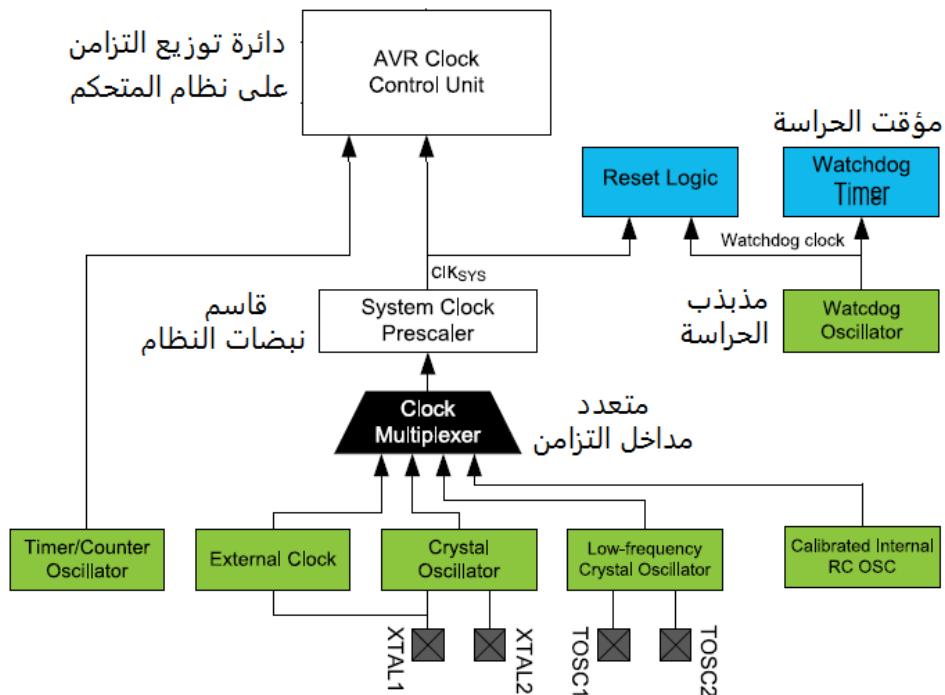
عند التعامل مع الشرائح avr كما جاءت من المصنع وبدون أي تعديل عليها يكون ذلك بالاعتماد على نبضات التزامن المبنية بداخلها والتي يبلغ ترددتها 1 ميجاهرتز. وهذا بالطبع يكون كافياً للكثير من التطبيقات، ولكن بفرض أننا نريد أن نعمل عند تزامن أعلى من ذلك ومن خارج الشريحة فماذا نفعل؟

بعد أن نكتب البرنامج ويتم حرقه على شريحة المتحكم، من الممكن لأي مستخدم آخر أن يأخذ الشريحة وبضعها على جهاز القراءة ويقوم بنسخ البرنامج الذي يعتبر حق فكري لك، فكيف نحمي برامجنا من هذا النسخ غير الشرعي؟ هذين السؤالين وغيرها سنجيب عليهما من خلال دراستنا لمجموعة براتات الفيوزات fuse bits الموجودة في كل متحكم، حيث من خلال وضع أي فيوز من هذه الفيوزات بصفر 0 (فعال)، أو بوحدة 1 (غير فعال) يمكن التحكم في الكثير من أدوات المتحكم. هناك برامج خاصة تمكّنك من التعامل مع هذه الفيوزات. نذكر هنا من أنه يجب أن يكون التعامل مع هذه الفيوزات بمتنه الحذر لأنه من الممكن أن تضع هذه الفيوزات في وضع لا يمكن الرجوع منه وبذلك تخسر شريحة المتحكم بالكامل وللأبد. ولذلك فإنه إذا لم تكن هناك حاجة للتعامل مع هذه الفيوزات فيفضل البقاء عليها كما هي، وكما جاءت من المصنع.

هذه الفيوزات هي في الحقيقة عبارة مجموعه من بآيات ذاكرة القراءة فقط القابلة للقراءة والكتابة EEPROM، والتي من مميزاتها أنها لا تفقد محتواها بانقطاع القدرة عنها، وهذا بالطبع ضروري لوضع هذه الفيوزات. هذه البآيات يكون لها عنوان خاص بكل منها مختلفا تماماً عن عناوين الذاكرة EEPROM الأخرى الموجودة في المتحكم والتي سبق شرحها، وهذه العناوين بالطبع تختلف من متحكم لآخر.

## ٢-١٤ مصادر نبضات تزامن المتحكم

شكل ١-١٤ يبين المصادر المختلفة لنبضات التزامن في المتحكم في atmega328 وهو نفس المصادر في الإصدارات الأخرى من المتحكمات avr مع اختلافات بسيطة.



شكل ١-١٤ المصادر المختلفة لنبضات التزامن في المتحكم atmega328

المرجعات الخضراء في شكل ١-١٤ توضح المصادر المختلفة لنبضات التزامن الداخلة للمتحكم، وهي ستة مصادر. أربع مصادر منها تدخل على متعدد المداخل ليختار أحدها عن طريق وضع باتاً معينة في أحد المسجلات سنراه بعد قليل. النبضات الخارجية من متعدد المداخل تدخل على قاسم للنبضات يمكن اختيار نسبة القسمة له عن طريق وضع باتاً معينة في أحد المسجلات أيضا سنراه بعد قليل. النبضات الخارجية من القاسم هي نبضات التزامن المستخدمة في كل نظام المتحكم وهي التي رمزنا لها بالرمز  $f_{OSC}$  في أثناء شرحنا لكل ملحقات المتحكم التي تستخدم نبضات تزامن النظام مثل المحول التماثلي الرقمي ADC، والمؤقتات، ووحدة المعالجة المركزية CPU أيضا.

هناك مصادران لا يدخلان على متعدد المداخل وهما مصدر النبضات الخاص بكلب الحراسة watchdog، وهو مذبذب خاص بكلب الحراسة فقط ولا يستخدم في أي غرض آخر، وسيأتي الحديث عن كلب الحراسة بشيء من التفصيل فيما بعد. هناك أيضا مصدر نبضات خاص بالمؤقتات وهو يمثل النبضات الخارجية الدخالة لكل واحد من

المؤقتات الثلاثة التي درسناها من قبل (المؤقت 0، المؤقت 1، المؤقت 2) حيث يوجد طرف خاص بكل مؤقت تدخل منه النبضات المراد عدها عند استخدام المؤقت كعداد. هذه النبضات لا تدخل على القاسم ولكنها تدخل مباشرة على وحدة توزيع نبضات التزامن من أجل تكيفها للتزامن مع أحد نبضات التزامن قبل دخولها إلى المؤقت الخاص بها. اختيار مصدر نبضات التزامن التي سيعمل عندها المتحكم وأشياء أخرى يتم من خلال براتف الفيوزات الموجودة في ٣ بait تسمى بيات الفيوزات، كل بت من بيات هذه البايتات تقوم بوظيفة معينة عند تنشيطها أو برمجتها كما هو مصطلح عليه. الجزء التالي يبين تفاصيل هذه البايتات.

### ٤-٣ مسجل معامل تقسيم نبضات التزامن

شكل ٤-١٤ يبين بيات هذا المسجل حيث نلاحظ أنه مكون من ثمان بيات مستخدم منها خمسة فقط، ووظيفة هذه البتات كالتالي:

7	6	5	4	3	2	1	0
CLKPCE				CLKPS3	CLKPS2	CLKPS1	CLKPS0
R/W				R/W	R/W	R/W	R/W
0				x	x	x	x

شكل ٤-١٤ مسجل معامل تقسيم نبضات التزامن

**البت CLKPCE ٧:** بت تنشيط تغيير معامل التقسيم clock Prescaler Change Enable، يجب وضع هذه البت بوحدة حتى يمكن تغيير محتويات البتات من صفر حتى ٣ التي ستحدد بها مقدار معامل التقسيم.

CLKPS[3:0]	معامل تقسيم النبضات
0000	1
0001	2
0010	4
0011	8
0100	16
0101	32
0110	64
0111	128
1000	256
1111-1001	Reserved

شكل ٤-١٤ جدول معامل تقسيم نبضات التزامن

**البتات من صفر حتى ٣ CLKPS[3:0]:** يتم وضع هذه البتات بواحد وأصفار للحصول على معاملات تقسيم نبضات التزامن القادمة من المصدر، وقبل إدخالها على نظام المتحكم. هذا المعامل يتغير من ١ حتى ٢٥٦ كما هو موضح في الشكل ٤-١٤.

## ٤-٤ بaitات الفيوزات

توجد هذه الفيوزات موزعة على ثلاثة بaitات تسمى bait المخفضة (أو الأولى) low byte، و bait العالية (أو الثانية) high byte، و bait الممتدة (أو الثالثة) extended byte وتفاصيلها كالتالى:

### البait المخفضة :low byte

هذه bait مكونة من ٨ بت وظائفها كالتالى وكما هو موضح في شكل ٣-١٤ :

٧	٦	٥	٤	٣	٢	١	٠
CKDIV8	CKOUT	SUT1	SUTO	CKSEL3	CKSEL2	CKSEL1	CKSEL0
٠	١	١	٠	٠	٠	١	٠

**القيمة التلقائية**

شكل ٣-١٤ bait المخفضة (الأولى) في الفيوزات

**البait ٧ CKDIV8:** تأتى شريحة المتحكم atmega328 من المصنع وقد تمت برمجة فيوزاتها بحيث يتم اختيار المذبذب RC الداخلى وبحيث يعمل هذا المذبذب عند التردد ٨ ميجاهرتز. لذلك فإن البait CKDIV8 تأتى من المصنع أيضا مبرمجة programmed، أى تساوى صفر، بحيث يتم قسمة هذا التردد على ٨ عن طريق قاسم التردد، لذلك فإن المتحكم القادم من المصنع يعمل تلقائيا عند التردد واحد ميجاهرتز. وهذا هو الوضع التلقائى الذى نعمل عنده في الكثير من التطبيقات والذى نوصى بالعمل به إلا إذا كانت هناك حاجة لتغييره. الحاجة التى تستدعي تغيير هذا الوضع هى الحاجة مثلا لزمن توقيت صغير جدا لا يمكن الحصول عليه من أى واحد من المؤقتات مع هذا التردد.

**البait ٦ CKOUT:** عندما تكون هذه البait مبرمجة أى تساوى صفر أى نشطة، فإنها تسمح بإخراج نبضات التزامن على الطرف CLKO وهو الطرف ١٤ في شريحة المتحكم (أو الطرف PB0)، وإخراج وظيفة هذا الطرف كطرف من أطراف البوابة B. هذه النبضات الخارجية تساوى نبضات تزامن النظام، أى الناتجة بعد القاسم وهى تعمل سواء كان مصدر النبضات داخلى أو خارجى. بالطبع يمكن للمستخدم أن يستخدم هذه النبضات فى أى أغراض خارج شريحة المتحكم.

**البنا ٤ و ٥ [SUT[1:0]]:** هذه البنا خاصه بتحديد زمن الاستعداد للبدأ Start Up Time, SUT. مع بداية تشغيل مصدر القدرة VCC الخاص بالمحكم، تبدأ نبضات التزامن سواء كانت داخلية أو خارجية في العمل، ولكن ما يحدث هو أن كل من مصدر القدرة وكذلك نبضات التزامن تأخذ بعض الوقت حتى تصل إلى حالة الاستقرار التي يمكن للمتحكم عندها أن يعمل. لذلك لابد مع بداية تشغيل المتحكم سواء بعد إعادة الوضع Reset أو تشغيل مصدر القدرة أن يتم الانتظار بعض الوقت قبل أن يبدأ المتحكم في العمل وإجراء الحسابات، ومقدار هذا الوقت يتوقف على كل من مقدار نبضات التزامن المختارة ومقدار مصدر القدرة الذي يعمل عنده المتحكم. يدخل في هذه اللعبة أيضا وحدة مراقبة مصدر القدرة Brown out detector, BOD. لذلك فإن البنا [SUT[1:0]] مع بنا اختيار نبضات التزامن التي سنشرحها في الجزء التالي مع وحدة مراقبة مصدر القدرة كلها تعمل على تحديد مقدار الزمن الذي سيتم انتظاره وعدد نبضات التزامن التي يجب انتظار مرورها (أو إهمالها) قبل أن يبدأ المتحكم في العمل. عدد هذه النبضات قد يكون قليلاً (حوالى ٦ نبضات)، أو كبيراً ( يصل إلى ٣٢ كيلونبضة). دليل شريحة المحكم atmega328 يقدم قيم مقترنة للبنات SUT وبنا اختيار مصدر الشريحة وزمن التأخير وعدد نبضات التزامن التي يجب أن تمر في الجدول ٤-١٣ في هذا الدليل، والذي لم نجد حاجة في إعادة نسخه هنا. من هذا الجدول سنلاحظ أن أسوأ الأحوال هي زمن تأخير مقداره ٦٥ ميلي ثانية زائد ٤ نبضة تزامن ويتم ذلك بوضع البنا SUT[1:0]=11 والبت CKSEL[0]=1 وهذا في حالة استخدام مذبذب بللورة crystal oscillator مع مصدر قدرة ذو زمن استقرار (زمن الارتفاع من الصفر لقيمه الفعالة) كبير. الحالة الثانية عند وضع البنا SUT[1:0]=00 والبت CKSEL[0]=1 وهذا في حالة استخدام مذبذب سيراميكي ceramic oscillator مع مصدر قدرة ذو زمن استقرار (زمن الارتفاع من الصفر لقيمه الفعالة) كبير. وهناك في الجدول حالات أخرى كثيرة بالطبع.

Device Clocking Option	مصدر النبضات المختار	CKSEL[3:0]
Low Power Crystal Oscillator	مذبذب بللوري منخفض القدرة	1111 - 1000
Full Swing Crystal Oscillator	مذبذب بللوري عالي القدرة	0111 - 0110
Low Frequency Crystal Oscillator	مذبذب بللوري منخفض التردد	0101 - 0100
Internal 128kHz RC Oscillator	مذبذب داخلي 128 كيلوهertz	0011
Calibrated Internal RC Oscillator	مذبذب داخلي معاير 8 كيلوهertz	0010
External Clock	مصدر نبضات خارجي	0000
Reserved	غير مستخدم	0001

شكل ٤-١٤ مصادر نبضات التزامن المختلفة و اختيارها بالبنات CKSEL[3:0]

القيمة التلقائية للبتات [SUT[1:0] والتي تكون مبرمجة في المتحكم عند شراؤه من المصنع هي: ١٠=١٠:SUT]. ونحن نرى أيضاً أنه إذا لم تكن هناك ضرورة لتغيير هذه القيم التلقائية فلنصح ببقائها كما هي. وبالمناسبة فإنه من الجدول ٤-١٣ في دليل المتحكم فإن عدد نبضات الانتظار هي ٤١ نبضة زائد زمن انتظار مقداره صفر وهذا بالطبع عند استخدام المذبذب الداخلي التلقائي.

**البتات صفر و ١ و ٢ و ٣ [CKSEL[3:0]]:** هذه البتات هي المسئولة عن تحديد مصدر نبضات التزامن الموضحة في شكل ١-١٤ الذي سيتم اختياره عن طريق متعدد المداخل الموجود في هذا الشكل. تلقائياً هذه البتات موضوعة لاختيار مصدر النبضات الداخلي بالقيم التالية: CKSEL[3:0]=0010 كما في شكل ٢-١٤، وهذا هو الوضع الأكثر أماناً كما ذكرنا من قبل. شكل ٤-١٤ يبين المصادر المختلفة لنبضات التزامن وكيفية اختيارها بالبتات CKSEL[3:0].

Frequency Range [MHz]	التردد	CKSEL[3:1]
0.4 - 0.9		100
0.9 - 3.0		101
3.0 - 8.0		110
8.0 - 16.0		111

شكل ٥-١٤ ترددات المصدر الداخلي منخفض القدرة

**١- مذبذب بللوري داخلي منخفض القدرة:** هذا المذبذب يعطي نبضات ذات مقدار منخفض لذلك فإن قدرته تكون منخفضة وهذه النبضات غير مناسبة لتشغيل أي دوائر أخرى لأنها قدرتها والضوضاء التي قد تكون عليها. خرج هذا المذبذب يمكن استخدامه من على الطرفين XTAL1 و XTAL2 في شريحة المتحكم.

يمكن لهذا المذبذب أن يعمل عند ٤ ترددات على حسب البتات CKSEL[3:1] كما هو مبين في شكل ٤-١٤. البت CKSEL0 مع البتات SUT[1:0] يتم بها اختيار زمن التأخير وعدد النبضات حتى الوصول حالة الاستقرار كما في الجدول ٤-١٣ في دليل المتحكم كما أشرنا مسبقاً.

**٢- مذبذب بللوري داخلي عالي القدرة:** خرج هذا المذبذب يمكن استخدامه أيضاً من على الطرفين XTAL1 و XTAL2 في شريحة المتحكم. الإشارة الخارجية منه تكون كاملة التأرجح ولذلك تكون قدرتها عالية وهو يعمل عند الجهد ٢,٧ فولت حتى ٥,٥ فولت. يتم اختيار هذا المذبذب بوضع باتات الاختيار CKSEL[3:1]=011 وتردد المذبذب في هذه الحالة سيتراوح من ٤٠٠ ميجاهرتز حتى ٢٠ ميجاهرتز على حسب قيمة جهد القدرة.

البت CKSEL0 مع البتات [1:0] SUT يتم بها اختيار زمن التأخير وعدد النبضات حتى الوصول لحالة الاستقرار كما في الجدول ٦-١٣ في دليل المتحكم كما أشرنا مسبقاً.

**٣- المذبذب البللوري الداخلي منخفض التردد:** هذا المذبذب يعمل عند تردد واحد وهو ٣٢،٧٦٨ كيلوهرتز وهو منخفض القدرة. يمكن اختياره بوضع البتات CKSEL[3:0]=0100 أو CKSEL[3:0]=0101 على حسب عدد النبضات المطلوبة حتى الوصول إلى حالة الاستقرار.

**٤- مذبذب داخلي RC ١٢٨ كيلوهرتز:** بوضع البتات CKSEL[3:0]=0011 يتم اختيار مذبذب داخلي بتردد ثابت ١٢٨ كيلوهرتز منخفض القدرة.

**٥- المذبذب الداخلي المعاير ٨ ميجا赫رتز:** وهو كما قلنا الاختيار التلقائي عند شراء المتحكم من المصنع ويتم اختياره بوضع البتات CKSEL[3:0]=0010.

**٦- مصدر نبضات خارجي:** يتم هذا الاختيار بوضع البتات CKSEL[3:0]=0000، ويتم توصيل هذه النبضات على الطرف EXTCLK، ويمكن للمتحكم أن يعمل في هذه الحالة عند ترددات من صفر حتى ٢٠ ميجا赫رتز. لاحظ أن هذا المصدر ليس بللورة ولكنه يكون مولد نبضات يعطي نبضات تزامن جاهزة بالتردد المطلوب. بذلك تكون قد تعرفنا على جميع برات البایت الأولى (المنخفضة low byte) من بايتات الفيوزات الثلاثة.

### البايت العالية :high byte

شكل ٤-٦ يبين برات هذه البايت حيث نلاحظ أن القيمة الابتدائية (القادمة من المصنع) هي 11011001 وأما وظيفة كل برت من هذه البتات فستكون كما يلى:

7	6	5	4	3	2	1	0
RSTDISBL	DWEN	SPIEN	WDTON	EESAVE	BOOTSZ1	BOOTSZ0	BOOTRST

القيم الابتدائية

شكل ٤-٦ البايت العالية (الثانية) في الفيوزات

**البت ٧ RSTDISBL:** من ضمن وظائف الطرف ١ (PC6) لشريحة المتحكم أنها يمكن استخدامها كطرف إعادة وضع reset للمتحكم، حيث بوضع نبضة تساوى صفر ملدة أكبر من زمن نبضة التزامن يحدث إعادة وضع للمتحكم. وهذا هو الوضع التلقائي للمتحكم عندما يأتي من المصنع. يمكن إبطال هذه الخاصية (خاصية إعادة وضع المتحكم) عن طريق برمجة هذه البت (وضعها تساوى صفر). في هذه الحالة يستخدم هذا الطرف كطرف حقيقي للبوابة C. أيضا لا ينصح هنا بعمل ذلك إلا إذا كنت فعلاً تحتاج لهذا الطرف كطرف دخل/خرج.

بمذه المناسبة فإن هناك أكثر من طريقة يتم من خلالها إعادة وضع المتحكم وهي:

- إعادة وضع لنقص جهد القدرة، حيث يحدث إعادة لوضع المتحكم طالما أن جهد القدرة أقل من قيمة جهدية معينة وهذا يحدث عادة عند بداية تشغيل مصدر القدرة للنظام.
- إعادة وضع من الخارج، وهو الطرف Reset على الطرف ١ للمتحكم والذي يمكن إلغاء وظيفته عن طريق البت .RSTDISBL
- إعادة وضع من كلب الحراسة، حيث عند انقضاء المدة المحددة ل الكلب الحراسة Watch dog timer يحدث إعادة وضع للمتحكم.
- إعادة وضع من نظام مراقبة مصدر القدرة، نظام مراقبة جهد القدرة Brown out detector هو نظام أو دائرة داخل المتحكم تقوم بمراقبة الجهد Vcc بحيث إذا نقص عن جهد تشبع معين يقوم هذا النظام بإعادة وضع المتحكم، بشرط أن يكون هذا النظام مفعلاً بالطبع.

**البت ٦ DWEN:** تنشيط سلك التصحيح Debug Wire Enable، عند تنشيط هذه البت (أى جعلها مبرمجة أو تساوى صفر) فإنه يتم تنشيط نظام التصحيح في المتحكم. نظام سلك التصحيح هو بروتوكول اتصال تتبعى مع شريحة المتحكم مثل بروتوكول JTAG وبروتوكولات أخرى. عند تنشيط هذه البت يتم التواصل مع المتحكم من خلال طرف إعادة الوضع (الطرف ١) حيث تكون خاصية إعادة الوضع غير مفعولة في هذه الحالة. من خلال هذه الطرف يمكن القراءة والكتابة من كل أجزاء الذاكرة، ووضع علامات توقف في البرنامج، وتنفيذ البرنامج بنظام الخطوة الواحدة، وغير ذلك من عمليات التصحيح المعروفة. حيث أنها لن تحرق البرنامج على شريحة المتحكم إلا بعد التأكد من صحته، وسيتم استخدام برنامج الأتمت استديو في إجراء هذه التصحيحات فإنه يفضل ترك هذه البت على حالتها التلقائية وهي الحالة غير المبرمجة (تساوى ١).

**البت ٥ SPIEN:** تنشيط الاتصالات المتتالية من خلال البروتوكول SPI الذى سبق شرحه. الوضع التلقائى لهذه البت أنها مبرمجة (أى نشطة أى تساوى صفر) وبالتالي فإن الاتصالات المتتالية تكون مسموحة بها. في أثناء تطوير مشروعك أو برنامجك ستحتاج للكتابة في الشريحة القراءة منها من خلال هذا البروتوكول، لذلك يستحسن عدم تغيير حالة هذه البت عن الحالة التلقائية. فقط عند الانتهاء من تطوير مشروعك وتريد حمايته بمنع أى تسجيل آخر في المتحكم أو القراءة منه، فقط في هذه الحالة ضع هذه البت على الوضع غير المبرمج (أى تساوى واحد).

**البت ٤ WDON:** تفعيل كلب الحراسة Watch Dog Timer ON، هذه البت تأتى تلقائياً غير مبرمجة (تساوى واحد)، وعند برمجتها (وضعها بصفر) يصبح كلب الحراسة نشط بحيث سيقوم بعمل reset أو مقاطعة للمتحكم عند مرور زمن سابق التحديد ولم يتم إعادة وضع مؤقت الحراسة. أيضاً إذا لم تكن هناك حاجة لتفعيل كلب الحراسة فلا داعي لتغييرها عن الوضع التلقائي. كلب الحراسة WDT يتكون من مذبذب خاص بتردد ١٢٨ كيلوهertz يتم إدخاله على معامل قسمة من ٢ حتى ١٠٢٤ من خلال بتات مسجل تحكم خاص بتشغيل هذا النظام. عند نفاد الوقت المحدد لهذا النظام يقوم المؤقت بعمل شيء من ثلاثة، أما أن يقوم بعمل إعادة وضع للمتحكم، أو مقاطعة للمتحكم، أو بعمل الاثنين معاً (إعادة وضع وعمل مقاطعة). تفاصيل ذلك موجودة في دليل المتحكم (أفضل مرجع يمكن الرجوع إليه في هذا الموضوع) ولن نخوض في تفاصيلها هنا لأنها نادراً ما تستخدم.

**البت ٣ EESAVE:** بت الحفاظ على محتويات الذاكرة EEPROM. كما نعلم فإن هذه الذاكرة تستخدم لتخزين البيانات المهمة فقط ولا تستخدم لتخزين البرنامج. هناك أوامر يمكن بها مسح كل ذاكرة المتحكم بما فيها هذه الذاكرة EEPROM إذا كانت هذه البت غير مبرمجة (تساوى واحد) وهو الوضع التلقائي لها. عند برمجة هذه البت أو تفعيلها (وضعها تساوى صفر)، فإنه إذا تم مسح الذاكرة فسيتم الحفاظ على محتويات هذه الذاكرة EEPROM ولن تمسح، وهذه بالطبع ميزة عظيمة جداً تفيد في الكثير من الأحوال.

**البت ٢ و ١ [1:0]: BOOTSZ**: الذاكرة اللحظية flash memory تكون في العادة مقسمة إلى قسمين: قسم خاص ببرامج التطبيقات، وقسم خاص ببرنامج تحميل برامح البدأ Boot Loader Section، BLS. البرامح أو الأكواد المكتوبة في قسم برامح التطبيقات هي البرامح العادية التي يتم تنفيذها عن طريق المستخدم وهي برامح التطبيقات الشائعة والمعروفة. برنامج تحميل برامح البدأ يتم كتابتها في الجزء الثاني من الذاكرة اللحظية BLS. هذه البرامح تقوم في

العادة بتعريف أى ملحقات موصلة على المتحكم مثلما يفعل النظام BIOS في نظم الحاسبات، أو برنامج البدأ في نظم الأردوينو، حيث بعد تحميل هذا البرنامج تنتقل عملية التنفيذ إلى جزء الذاكرة الخاص بالتطبيقات.

- البتات BOOTSZ1 و BOOTSZ0 تستخدم لتحديد حجم كل قسم من قسمى الذاكرة. شكل ٤-٧ يبين جدولًا يحدد مقدار وعنوان بداية كل قسم من قسمى الذاكرة اللحظية.

BOOTSZ1	BOOTSZ0	Boot Size	Pages	Application Flash Section	Boot Loader Flash Section	End Application Section	Boot Reset Address (Start Boot Loader Section)
1	1	256 words	4	0x0000 - 0x3EFF	0x3F00 - 0x3FFF	0x3EFF	0x3F00
1	0	512 words	8	0x0000 - 0x3DFF	0x3E00 - 0x3FFF	0x3DFF	0x3E00
0	1	1024 words	16	0x0000 - 0x3BFF	0x3C00 - 0x3FFF	0x3BFF	0x3C00
0	0	2048 words	32	0x0000 - 0x37FF	0x3800 - 0x3FFF	0x37FF	0x3800

شكل ٤-٧ مقدار وعنوان بداية كل قسم من قسمى الذاكرة (التطبيقات أو برمج البدأ)

من هذا الجدول نلاحظ أن أقل كمية ذاكرة لبرام吉 البدأ هي ٢٥٦ وورد، وأكبر كمية هي ٢٠٤٨ وورد. نلاحظ أيضاً أن ذاكرة التطبيقات تبدأ من العنوان 0x0000 ويأتي في آخر الذاكرة اللحظية القسم الخاص ببرام吉 البدأ. لاحظ أن الوضع التلقائي لهذه البتات هو أنها تكون مبرمجة (تساوي أصفار) وبالتالي فإن مقدار ذاكرة البدأ سيكون ٢٠٤٨ وورد كما في شكل ٤-١.

**البت صفر BOOTRST:** هذه البت تكون تلقائياً غير مبرمجة (تساوي واحد)، وبالتالي فإن المتحكم مع بدأ تشغيله أو إعادة وضعه يبدأ التنفيذ من قسم الذاكرة الخاص بالتطبيقات. عند برمجة هذه البت (وضعها تسوى صفر) فإن المتحكم بعد إعادة وضعه reset أو عند توصيل القدرة له سيبدأ التنفيذ من قسم الذاكرة الخاص ببرام吉 البدأ. ولذلك فإنه إذا لم تكن مستخدماً لبرام吉 بدأ موضوع في هذا القسم من الذاكرة فلا داعي لبرمجة هذه البت وتترك على وضعها التلقائي.

بذلك تكون قد انتهينا من استعراض كل بتات البايت العالية (الثانية) من الفيوزات.

البايت الممتدة extended byte

تتكون هذه البايت من 8 بت مستخدم منها فقط أول ثلاثة بิตات. وظيفة هذه البتات هي:

**البت ٢ و ١ و صفر [2:0] BODLEVEL:** تستخدم هذه البتات الثلاثة لتحديد قيمة الجهد التشبعى الذى إذا نزل جهد مصدر القدرة عنه فإن نظام مراقبة القدرة Brown out detector سيعيد وضع المتحكم. شكل ٨-١٤ يبين رسمياً توضيحاً لهذه البايت، والجدول المبين في شكل ٩-١٤ يبين الشفرات الموضعية على هذه البتات في مقابل قيم الجهد التشبعى  $V_{BOT}$ ,  $V_{BOT}$



شكل ٨-١٤ البايت الممتدة في الفيوزات

لاحظ أن القيم التلقائية لهذه البتات الثلاثة هي أنها تكون غير مبرمجة (تساوي وحيد) وهذا يعني أن نظام مراقبة جهد مصدر القدرة يكون مخدداً أو لا يعمل كما في جدول الشكل ٩-١٤.

BODLEVEL [2:0] Fuses	Min. $V_{BOT}$	Typ. $V_{BOT}$	Max $V_{BOT}$	Units
111	نظام مراقبة مصدر القدرة لا يعمل			
110	1.7	1.8	2.0	V
101	2.5	2.7	2.9	
100	4.1	4.3	4.5	
011	Reserved			
010	محجوزة			
001				
000				

شكل ٩-١٤ القيم المختلفة للجهد التشبعى لنظام مراقبة جهد مصدر القدرة

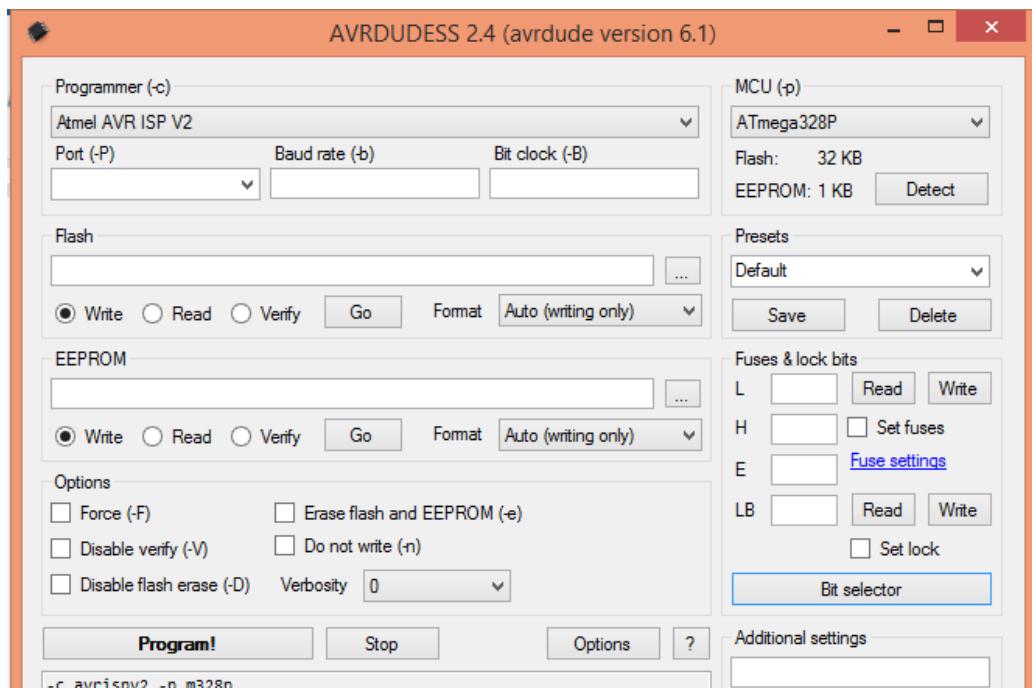
بذلك تكون قد انتهينا من باياتات الفيوزات الثلاثة الموجودة في المتحكم atmega328 وبالطبع قد يكون بعض التغييرات البسيطة في الإصدارات الأخرى من المتحكمات avr. قبل أن نترك هذا المجال هناك بait رابعة وهي الخاصة

بتؤمن المتحكم أو إغلاقه وهي بait الإغلاق Lock Byte. هذه البايت تحتوى 8 بتات مستخدم منها ستة بتات يمكن برمجة كل منها لعمل الآتى:

- إغلاق جزء الذاكرة الخاص بالتطبيقات فقط
- إغلاق جزء الذاكرة الخاص بتحميل برامج البدأ boot loader فقط
- إغلاق كل من جزئي الذاكرة (التطبيقات وتحميل برامج البدأ)
- إتاحة كل من جزئي الذاكرة

هذه البايت تأتى غير مبرمجة، بمعنى أن كل بتاتها تساوى واحد، وهذا هو الوضع التلقائى لها، وينصح دائماً بعدم برمجة هذا الجزء إلا عند الضرورة وذلك لحماية بعض أجزاء ذاكرة المتحكم من النسخ أو القراءة.

الخطوة الأخيرة في هذا الفصل وبعد أن عرفنا وظيفة كل فيوز من هذه الفيوزات ستكون هي حرق هذه الفيوزات على شريحة المتحكم. هذه العملية سهلة جداً من خلال برنامج avrdudeess المتاح مجاناً على الإنترنت. بعد تنزيل هذا البرنامج على الحاسب الخاص بك وتثبيته يمكنك البدأ في حرق الفيوزات من خلال خطوتين فقط.



شكل ١٤-١٠ شاشة اختيار المتحكم وأداة البرمجة في برنامج avrdudeess

الخطوة الأولى هي النقر على أيقونة البرنامج حيث سيفتح لك شاشة كامووضحة في شكل ١٤-١٠. من هذه الشاشة إختر المتحكم الذي تتعامل معه من مربع في أعلى يمين الشاشة وهو في حالتنا هذه المتحكم atmega328. إختر أيضا دائرة البرمجة التي تستخدمها في برمجة المتحكم في مربع في أعلى يسار الشاشة. بعد ذلك أنقر على الزر Bit Selector على يمين الشاشة حيث سيتم فتح شاشة جديدة تحتوي كل بيانات الفيوزات التي من الممكن برمجتها كما في الشكل ١٤-١١.

الخطوة الثانية: في هذه الشاشة الجديدة قم بالنقر على الفيوزات التي تريد تغييرها ثم انقر على الزر OK، حيث عندها سيتم حرق الفيوزات بالوضع الذي اخترته على شريحة المتحكم، ومن خلال المبرمج الذي اخترته.



شكل ١٤-١١ شاشة اختيار الفيوزات في برنامج avrdudeess

المفروض أن خطوة حرق الفيوزات تكون هي آخر خطوات برمجة المتحكم (إذا كانت هناك حاجة ضرورية لها كما أشرنا مسبقاً)، حيث بعد ذلك يمكن وضع شريحة المتحكم في مكانها في المشروع.

تذكر أن هناك العديد من البرامج التي يمكن استخدامها لحرق الفيوزات ومعظمها متاح على الإنترنت مجاناً، ولكننا نعتقد أن أبسطها هو هذا البرنامج avrdudeess. تذكر أيضاً أن هناك إصدارات قديمة من هذا البرنامج تسمى avrdude وليس لها واجهة للتعامل مع المستخدم ولكن التعامل معها يكون من خلال أوامر تكتب في شاشة يفتحها البرنامج. لذلك نوصى بالتعامل مع برنامج avrdudeess لبساطته وسهولته.

## ملخص الفصل

لقد تناول هذا الفصل الخطوة الأخيرة من خطوات تصميم أي نظام مدمج والتي تكون عبارة عن حرق فيوزات الحماية للبرنامج المكتوب على شريحة المتحكم من النسخ أو التلاعب به. تم ذلك من خلال تعريفنا على كل واحد من هذه الفيوزات ووظيفتها كل منها والوضع التقائي لها. وأوصينا في الفصل، ونوصي هنا أيضاً، بأنه إذا لم تكن هناك ضرورة لتغيير الوضع التقائي لهذه الفيوزات فيجب أن تترك كما هي. في النهاية قدم الفصل كيفية حرق هذه الفيوزات (إذا حدث وقمت بتغييرها) على شريحة المتحكم من خلال برنامج avrdude المتاح على الإنترنت من خلال خطوتين اثنين فقط.

## الملحق الأول

ملف الأول، c.، من مكتبة الشاشة LCDlib.c، وهو الملف LCDlib.c. يمكنك نسخه كما هو ووضعه في ملف في الأتميل استديو مباشرة.

```
/*
 * LCDlib.c
 *
 * Created: 6/22/2017 4:55:36 PM
 * Author: Mohamed Eladawy
 */
#ifndef F_CPU
#define F_CPU 10000000ul
#endif
```

```
#include <avr/io.h>
#include <util/delay.h>
#include "lcdlib.h"
```

سنفترض هنا أن مسار بيانات الشاشة سيوصل على البوابة B

//

وأن خطوط التحكم الثلاثة RS و WR و EN سيتم توصيلها على أول ثلاث خطوط من البوابة C

//

```
define databus_direction DDRB // مسجل الاتجاه لتحديد اتجاه بوابة مسار بيانات الشاشة
define controlbus_direction DDRC // مسجل الاتجاه لتحديد اتجاه مسار التحكم للشاشة
```

```
define databus PORTB // تحديد البوابة التي سيتم مسار البيانات للشاشة عليها
define control_bus PORTC // تحديد البوابة التي سيتم توصيل خطوط التحكم عليها
```

```
#define rs 0 // RS
#define rw 1 // RW
#define en 2 // EN
```

رقم البت الذى سيوصل عليها خط التحكم

/\* بفرض أن الشاشة المستخدمة تحتوى على سطرين فقط كل منهما 16 حرفا \*/

```
#define LCDMaxLines 2
#define LCDMaxChars 16
#define LineOne 0x80
#define LineTwo 0xc0
```

```
#define BlankSpace ' '
/* هذه الدالة خاصة بتجهيز الشاشة لتعمل بطريقة 8 بت */
```

```

void LCD_Init()
{
    _delay_ms(50);
    تحديد البوابة B والبوابة C ليكونا بوابات خرج //
    databus_direction = 0xff;
    controlbus_direction = 0xff;
    LCD_CmdWrite(0x38); // نداء على هذه الدالة بهذه الشفرة للتعريف بشاشة من سطرين وكل حرف
    مصغوفة ٧x٥
    LCD_CmdWrite(0x0E); // نداء على هذه الدالة بهذه الشفرة لتشغيل الشاشة وإظهار دليل الكتابة //
    LCD_CmdWrite(0x01); // مسح الشاشة من أي حروف //
    LCD_CmdWrite(0x80); // وضع الدليل عند أول مكان في أول سطر //
}

هذه الدالة ترسل أمر إلى الشاشة // void LCD_CmdWrite( char cmd )
{
    وضع الأمر على مسار البيانات // databus=cmd;
    التسجيل في مسجل الأوامر // control_bus &=~(1<<rs);
    تنشيط عملية الكتابة // control_bus &=~(1<<rw);
    جعل خط التنشيط يساوى واحد // control_bus |=1<<en;
    _delay_ms(1);
    جعل خط التنشيط يساوى صفر // control_bus &=~(1<<en);
    _delay_ms(1);
}

هذه الدالة ترسل حرف للشاشة لإظهاره // void LCD_DataWrite( char dat )
{
    وضع الحرف على مسار البيانات // databus=dat;
    التسجيل في مسجل البيانات // control_bus |=1<<rs;
    تنشيط عملية الكتابة // control_bus &=~(1<<rw);
    جعل خط التنشيط يساوى واحد // control_bus |=1<<en;
    _delay_ms(1);
    جعل خط التنشيط يساوى صفر // control_bus &=~(1<<en);
    _delay_ms(1);
}

هذه الدالة تعرض سلسلة أحرف // void LCD_DisplayString(char *string_ptr)
// يتم إرسال الأحرف تلو الآخر إلى دالة عرض البيانات وإنفاص عدد بمقدار ١ إلى أن يصبح صفراء
{
    while(*string_ptr)
        LCD_DataWrite(*string_ptr++);
}

```

```
}
```

هذه الدالة تعرض أى رقم صحيح من صفر حتى // ٦٥٥٣٦ حيث يتم التعامل خانة بخانة وتحويلها إلى النظام الأسکى بإضافة الرقم ٣٠ ستعشرى

```
{
    LCD_DataWrite((num/10000)+0x30);
    num=num%10000;
    LCD_DataWrite((num/1000)+0x30);
    num=num%1000;
    LCD_DataWrite((num/100)+0x30);
    num=num%100;
    LCD_DataWrite((num/10)+0x30);
    LCD_DataWrite((num%10)+0x30);
}
```

الملف الثاني من ملفات المكتبة LCDlib.h وهو الملف

```
/*
 * LCDlib.h
 *
 * Created: 6/24/2017 6:56:37 AM
 * Author: Mohamed Eladawy
 */
```

```
#ifndef LCDLIB_H_
#define LCDLIB_H_

#define databus_direction DDRB
#define controlbus_direction DDRC

#define databus PORTB
#define control_bus PORTC

#define rs 0
#define rw 1
#define en 2

/* 16x2 LCD Specification */
#define LCDMaxLines 2
#define LCDMaxChars 16
```

```
#define LineOne 0x80
#define LineTwo 0xc0

#define BlankSpace ' '

void LCD_Init();
void LCD_Clear();
void LCD_GoToLineOne();
void LCD_GoToLineTwo();
void LCD_GoToXY(char row, char col);
void LCD_CmdWrite( char cmd);
void LCD_DataWrite( char dat);
void LCD_DisplayString(char *string_ptr); void
LCD_DisplayNumber(unsigned int num);

#endif /* LCDLIB_H_ */
```

## الملحق الثاني

ملف الأول، c..، من مكتبة مصفوفة المفاتيح Keypadlib.c، وهو الملف Keypadlib.c. يمكنك نسخه كما هو ووضعه في ملف في الأقل استديو مباشرة.

```

/*
 * Kepadlib.c
 *
 * Created: 6/27/2017 12:13:19 PM
 * Author: Administrator
 */
#ifndef F_CPU
#define F_CPU 10000000ul
#endif

#include <avr/io.h>
#include <util/delay.h>
#include "Keypadlib.h"
#define RowColDirection DDRD //Data Direction Configuration for keypad
#define ROW PORTD           //higher four bits of PORTD are used as ROWs
#define COL PIND            //Lower four bits of PORTD are used as COLs

void KEYPAD_Init()
{
    DDRC=0xff;
    RowColDirection=0xf0;   // ROW lines are configured as Output.
                           // Column Lines are configured as Input.
}

void WaitForKeyRelease() //This function waits till the previous key is
released.
{
    unsigned char key;
    do
    {
        ROW=0x0f;      // Pull the ROW lines to low and Column high low.
        key=COL & 0x07; // Read the Columns, to check the key press
    }while(key!=0x07); // Wait till the Key is released,
                       // If no Key is pressed, Column lines will remain high (0x0f)
}

void WaitForKeyPress() // This function waits till a new key is pressed.
{
    unsigned char key;
    do

```

```

{
    do
    {
        ROW=0x07; // Pull the ROW lines to low and Column lines high.
        key=COL & 0x07; // Read the Columns, to check the key press
    }while(key==0x07); // Wait till the Key is pressed,
    // if a Key is pressed the corresponding Column line go low

    _delay_ms(1); // Wait for some time(debounce Time);

    ROW=0x07; // After debounce time, perform the above operation
    key=COL & 0x07; // to ensure the Key press.

}while(key==0x07);
}

unsigned char ScanKey() //This function scans all the rows to decode the key
pressed.
{
    unsigned char ScanKey = 0xe0,i, key;

    for(i=0;i<0x04;i++) // Scan All the 4-Rows for key press
    {
        ROW=ScanKey + 0x07;// Select 1-Row at a time for Scanning the Key
        key=COL & 0x07; // Read the Column, for key press

        if(key!= 0x07)// If the KEY press is detected for the selected
        break; // ROW then stop Scanning,

        ScanKey=(ScanKey<<1)+ 0x10; // Rotate the ScanKey to SCAN the
remaining Rows
    }

    key = key + (ScanKey & 0xf7); // Return the row and COL status to
decode the key

    return(key);
}

unsigned char GetKey()//This function waits till a key is pressed and returns
its BCD Value
{
    unsigned char key;

    WaitForKeyRelease(); // Wait for the previous key release
    _delay_ms(1);
}

```

```

WaitForKeyPress();      // Wait for the new key press
key = ScanKey();
switch(key)    // Decode the key
{
    case 0xe6: PORTC=0x01; break;
    case 0xe5: PORTC=0x02; break;
    case 0xe3: PORTC=0x03; break;
    case 0xd6: PORTC=0x04; break;
    case 0xd5: PORTC=0x05; break;
    case 0xd3: PORTC=0x06; break;
    case 0xb6: PORTC=0x07; break;
    case 0xb5: PORTC=0x08; break;
    case 0xb3: PORTC=0x09; break;
    case 0x76: PORTC=0x0A; break;
    case 0x75: PORTC=0x00; break;
    case 0x73: PORTC=0x0b; break;
    default:   PORTC=0x00;
}
}

unsigned char GetKeyForLCD()//This function waits till a key is pressed and returns its
ASCII Value to display it on an LCD
{
    unsigned char key;

    WaitForKeyRelease();    // Wait for the previous key release
    _delay_ms(1);

    WaitForKeyPress();      // Wait for the new key press
    key = ScanKey();
    switch(key)    // Decode the key
    {
        case 0xe6: key='1'; break;
        case 0xe5: key='2'; break;
        case 0xe3: key='3'; break;
        case 0xd6: key='4'; break;
        case 0xd5: key='5'; break;
        case 0xd3: key='6'; break;
        case 0xb6: key='7'; break;
        case 0xb5: key='8'; break;
        case 0xb3: key='9'; break;
        case 0x76: key='A'; break;
        case 0x75: key='0'; break;
        case 0x73: key='B'; break;
        default:   key='Z';
    }
    return(key);
}

```

الملف الثاني من ملفات المكتبة Keypadlib.h وهو الملف

```
/*
 * Keypadlib.h
 *
 * Created: 6/27/2017 1:04:14 PM
 * Author: Administrator
 */

#ifndef KEYPADLIB_H_
#define KEYPADLIB_H_

#define RowColDirection DDRD //Data Direction Configuration for keypad
#define ROW PORTD           //higher four bits of PORTD are used as ROWs
#define COL PIND            //Lower four bits of PORTD are used as COLs

void KEYPAD_Init();
void WaitForKeyRelease(); //This function waits till the previous key
is released.
void WaitForKeyPress(); // This function waits till a new key is
pressed.
unsigned char ScanKey(); //This function scans all the rows to decode
the key pressed.
unsigned char GetKey(); //This function waits till a key is pressed and
returns its BCD Value
unsigned char GetKeyForLCD();

#endif /* KEYPADLIB_H_ */
```



# القاموس

# Dictionary

**A****Access time**

زمن الاتصال، بشريحة ذاكرة. وهو الزمن من لحظة وضع إشارة عنوان معين إلى لحظة استلام الخرج على خطوط البيانات من هذه الشريحة.

**Adder**

مجمع، يجمع رقمان. منه المجمع الرقمي الذي يجمع رقمين ثنائيين، والمجمع الانسيابي أو التماثلي الذي يجمع إشارتين انسيابيتين مثل مكبر العمليات.

**Address**

عنوان. إشارة أو رقم ثالثي يحدد عنوان بait معينة في نظام ذاكرة معين. عدد بتات هذا العنوان يحدد كمية الذاكرة التي يمكن التعامل معها في هذا النظام.

**Address bus**

مسار العنوانين، مجموعة من أطراف المعالج تخرج عليها إشارة العنوانين الثنائية التي من خلالها يحدد المعالج العنوان المراد التعامل معه. هذه الإشارة تكون دائماً خارجة من لمعالج.

**Amplitude**

مقدار، وتطلق على مقدار الإشارة. وهو أحد السمات المهمة التي تعرف بها أي إشارة.

**Analog**

انسيابي، أو تماثلي، أو مستمر، أو غير متقطع مثل تغير درجة الحرارة على مدار اليوم التي يمكنها أن تأخذ مالانهاية من القيم بين قيمتها الصغرى والعظمى.

**Analog to digital converter, ADC**

المحول التماثلي الرقمي ADC دخله إشارة تماثلية وخرجه إشارة رقمية مكونة من عدد من البتات، وهناك العديد من طرق التحويل التي يتوقف عليها زمن هذا التحويل.

**AND gate**

بوابة الآند AND، أو بوابة "و"، أو بوابة الضرب المنطقي. خرجها يساوى واحد في حالة واحدة فقط وهي عندما تكون كل دخولها تساوى واحد.

**Assembly language**

لغة التجميع، وهي لغة أوامرها تتكون من شفرات حرفية مختصرة مثل ADD و SUB، وكل معالج أو متحكم تكون له لغة التجميع الخاصة به، وهي أقرب ما يكون إلى لغة الآلة.

**Astable**

عديم الاستقرار، أو عديم الثبات، خرج يتردد باستمرار بين الواحد والصفر ولا يستقر على أي حالة منها.

**Asynchronous**

غير توافقى، لا يتغير بالتوافق مع نبضات تزامن معينة. يمكن تصنيف الدوائر الرقمية إلى توافقية وهى التي يتغير خرجها بالتوافق مع نبضات تزامن، وغير توافقية وهى التي لا تحتاج لنبضات تزامن تتوافق معها.

**B****Bidirectional**

ثنائي الاتجاه. مثلاً مسجل إزاحة ثنائي الاتجاه يمكن إزاحة بياته من اليمين لليسار أو العكس. أو مسار البيانات في المعالجات الذي يكون ثنائي الاتجاه حيث تكون الإشارة عليه خارجة من المعالج أو داخلة إليه.

**Binary**

ثنائي، Binary signal إشارة ذات مستويين، مستوى عالي (واحد) ومستوى منخفض (صفر). وهناك نظام العد الثنائي الذي له رقمان، صفر واحد.

**Binary Coded Decimal, BCD**

عشرى مكود ثنائياً، وضع الأرقام العشرية من صفر حتى تسعه في صورة أكواد ثنائية من أربع خانات.

**Bipolar**

ثنائي القطبية. Bipolar transistor ترانزستور مصنوع بتكنولوجيا القطبية الثنائية والتي تعنى التعامل مع حوامل شحنات سالبة وموجبة في نفس الترانزستور.

**Bistable**

ثنائي الاستقرار، دائرة لها حالتين من حالات الاستقرار.

**Bit**

الخانة في نظام العد الثنائي التي تكون واحد أو صفر.

**Boolean algebra**

الجبر البوليني، نسبة إلى عالم إنجلزي، وهو عبارة عن مجموعة قوانين جبرية خاصة بالتعامل مع المتغيرات المنطقية.

**Borrow**

استلاف من خانة تالية إلى الخانة الحالية في أثناء عمليات الطرح في كل نظم العد.

**Bounce**

إهتزاز. عند غلق أو فتح مفتاح ميكانيكي فإنه يحدث اهتزازات ميكانيكية غير مرغوب فيها، وهذه الاهتزازات تحدث ضوضاء كهربائية في صورة نبضات تؤثر على أداء الدوائر المنطقية.

**Buffer**

عازل أو فاصل. دائرة تستخدم لفصل الحمل عن الدائرة المغذية لها، وبذلك لا يؤثر الحمل العالى على أداء الدائرة المغذية. وقد يكون عازل رقمي أو تماثلى.

**Bus**

مسار، مجموعة من خطوط الاتصال بين عناصر نظام إلكترونى معين. مثلاً مسار العناوين يحمل إشارة العناوين بين شريحة المعالج وشريحة ذاكرة. ومسار البيانات الذى يحمل إشارة البيانات بين المعالج والذاكرة.

**Byte**

ثمانى بناة.

**C****Capacitor**

مكثف الشحنات، capacitance هي السعة الكهربية.

**Carry**

الحمل من خانة إلى خانة تالية فى أثناء عمليات الجمع.

**Central Processing Unit, CPU**

وحدة المعالجة المركزية، هي الوحدة التى تقوم بعمليات الحساب والتحكم والتزامن داخل أى حاسوب أو متتحكم.

**Clear**

تصفير، طرف غير توافقى يجعل الخرج صفر بدون توافق مع نبضات الساعة.

**Clock**

نبضات التزامن، أو الإطلاق. نبضات لها شكل معين يتزامن معها عمل نوع مهم من الدوائر المنطقية وهى الدوائر التتابعية أو الدوائر التوافقية.

**Code**

مجموعة من البناة تمثل شفرة لمعلومة معينة.

**Combinational logic circuit**

دائرة منطقية توافقية، دائرة مكونة من مجموعة من البوابات المنطقية الموصولة مع بعضها بحيث لا تحتوى على أى عنصر من عناصر الذاكرة مثل القلابات وما يعلوها. خرج هذه الدوائر يكون دالة فى الدخل فقط عند نفس اللحظة، ولا يعتمد على الخرج عند لحظات سابقة ولا تحتاج لنبضات تزامن لكي يتوافق الخرج معها.

**Commutative law**

فى بعض العمليات الحسابية والمنطقية لا يهم الطريقة التى ترتتب بها المتغيرات .  $x+y=y+x$  .

**Comparator**

مقارن، دائرة لمقارنة رقمين وتقرر إذا كانا متساوين أم أن أحدهما أكبر من الآخر، ويوجد المقارن الرقمي، والمقارن الانسيابي.

**Complement**

المتم، المتم الأحادى ones complement هو معكوس أى رقم ثالثي. المتم الثنائى twos complement هو معكوس الرقم الثنائى مضافاً إليه واحد المتم لأى رقم فى أى نظام عد هو حاصل طرح هذا الرقم من قاعدة هذا النظام.

**Counter**

عداد، بعد النبضات الداخلة له وهو العداد الرقمي.

**D****Data**

بيانات، فى العادة تكون ممثلة فى النظام الثنائى.

**Data bus**

مسار البيانات، مجموعة من أطراف المعالج تحمل إشارة البيانات الثنائية الداخلية أو الخارجية من المعالج.

**D flip flop**

قلاب له دخل واحد اسمه D حيث يصبح الخرج هو الدخل D بعد إعطاء نبضة التزامن.

**Debugging**

استخراج الأخطاء من أى برنامج.

**Decade**

دائرة تتميز بعشرة حالات. Decade counter عداد عشري له عشرة حالات.

**Decimal**

عشري. نظام العد العشري الذى له عشرة أرقام من صفر حتى تسعة.

**Decoder**

محل شفرة. دائرة رقمية تدخل لها شفرة رقمية فيحولها إلى صورة أخرى في الخرج. إذا كان عدد بناة شفرة الدخل هو  $n$  فإن محل الشفرة في هذه الحالة يكون له عدد  $2^n$  من المخارج يتم تنشيط أحدها على حسب شفرة الدخل.

**Decrement**

ينقص بمقدار واحد.

**Demultiplexer**

موزع، دائرة رقمية دخلها عبارة عن إشارة واحدة يتم توزيعها على مخارج الدائرة فى تتابع زمنى معين على حسب شفرة على خطوط خاصة لاختيار أحد هذه المخارج.

**Digital**

رقمي. الإشارة الرقمية هى إشارة مقطعة لها قيم محددة عند أزمنة محددة.

**Digit**

رقم، يمثل خانة معينة فى أحد أنظمة العد.

**Driver**

دافع تيار. يستخدم لدفع تيار عالي في الدوائر التي تحتاج لذلك.

**Dynamic RAM, DRAM**

ذاكرة اتصال عشوائي ديناميكية. تتميز بسرعة الاتصال ورخص الثمن ولكنها تحتاج لإعادة تسجيل محتوياتها كل 4 ميللي ثانية وإلا فإنها تفقد هذه المحتويات ووحدة بناؤها هو المكثف.

**E****Edge triggered flip flop**

قلاب يغير من حالة خرجه عند حافة نبضة التزامن سواء كانت الحافة الصاعدة (صفر إلى واحد) أو الحافة النازلة (واحد إلى صفر).

**EEPROM**

ذاكرة قراءة فقط يمكن برمجتها ومسحها كهربيا.

**Embedded system**

النظام الكامن عبارة عن تطبيق يحتوى شريحة واحدة على الأقل قابلة للبرمجة (مثل الميكروكونترولر)، وهذا التطبيق يستخدم عادة من قبل أشخاص قد لا يعلمون أن هذا النظام يقوم على ميكروكونترولر بداخله.

**EPROM**

ذاكرة قراءة فقط يمكن برمجتها بطرق خاصة ومسحها بالتعريض لأشعة فوق بنفسجية عالية الكافية.

**Enable**

تنشيط. طرف يستخدم لتنشيط خرج الدائرة المنطقية ثلاثة المنطق بحيث عندما يكون هذا الطرف غير فعال يكون خرج الدائرة عبارة عن مقاومة عالية.

**Encoder**

مولد الشفرة، دائرة تحول البيانات الداخلة إلى صورة مكودة أو مشفرة. المشفر الرقمي الذي له عدد من خطوط الدخل وعدد من خطوط الخرج، بحيث أنه عند تنشيط أحد خطوط الدخل فإنه يتم إعطاء شفرة لهذا الخط على كل خطوط الخرج.

**Exclusive NOR**

عملية منطقية على متغيرين تعطى صفرًا في حالة عدم تساوي المتغيرين.

**Exclusive OR**

عملية منطقية على متغيرين تعطى واحدًا في حالة عدم تساوي المتغيرين.

**F****Frequency**

التردد. عدد مرات التكرار في الثانية. عدد النبضات في الثانية. وحداتها هي الهرتز.

**Feedback**

التغذية المرتدة، وهي جزء من خرج أي دائرة يرجع أو يرتد إلى دخلها.

**Flip flop**

قلاب أو ناطط. دائرة منطقية ذات خرجين منتقبين كل منهما عكس الآخر. هناك أكثر من نوع منها على حسب الدخل، فهناك النوع JK والنوع D والنوع T وغيرها. هناك طرف تزامن للقلاب لا يتغير الخرج إلا عند إعطاء نبضة على هذا الطرف.

**Full Adder, FA**

مجموع كامل، دائرة تجمع 3 بت ويعطي مجموع وحمل للمرحلة التالية.

**G****Gate**

بوابة، دائرة لها مجموعة من المدخل وخرج واحد. يتم إجراء عملية منطقية تمثل هذه البوابة على المدخل ووضع نتيجة العملية على الخرج. هناك أنواع عديدة من البوابات.

**Glitch**

نتوء يظهر في المخطط الزمني وهو غير مرغوب فيه ومن الممكن أن يسبب مشاكل في تشغيل بعض الدوائر. انظر العادات الرقمية مثلاً.

**H****Half adder**

نصف مجموع، يجمع 2 بت فقط ويعطي مجموع وحمل للمرحلة التالية.

**Hexadecimal**

ستعشري. نظام العد الذي قاعدته 16 ويحتوى ستة عشر رقماً تبدأ بالصفر وتنتهي بالرقم F.

**Hold time**

زمن المسك، وهو الفترة الزمنية التي يجب أن يظل الدخل فيها مستقرًا بعد تطبيق الحافة المؤثرة لنبضة التزامن حتى يتغير الخرج بصورة مستقرة ومحددة.

**I****Increment**

الزيادة بمقدار واحد.

**In System Programming, ISP**

برمجة شريحة المتحكم وهى فى التطبيق أو النظام، دون الحاجة لفصلها من النظام من أجل برمجتها.

**Integrated Circuit, IC**

دائرة تكاملية. وهى نظام إلكترونى متكامل على شريحة واحدة لأداء وظيفة معينة.

**Interrupt**

المقاطعة، مقاطعة المسار الطبيعي لعملية تنفيذ البرنامج الأساسى إما من برنامج آخر، أو نتيجة حدوث أمر غير طبيعى فى البرنامج، مثل القسمة على الصفر، أو بسبب إشارةقادمة على أحد أطراف المتحكم تتنزد مثلاً بارتفاع درجة الحرارة.

**Interrupt service routine, ISR**

برنامج خدمة المقاطعة، عند حدوث المقاطعة من أي مصدر، يقفز المتحكم بعمليات التنفيذ من البرنامج الذى كان يقوم بتنفيذها وقت حدوث المقاطعة إلى برنامج آخر ينفذ الغرض من هذه المقاطعة. فإذا كانت المقاطعة مثلاً بسبب ارتفاع درجة الحرارة، يكون برنامج خدمة المقاطعة هو برنامج لتشغيل مبرد يعمل على تخفيض الحرارة.

**Inverter**

عاكس، بوابة عكس، خرجها عكس دخಲها.

**J****JK flip flop**

قلاب تم التغلب فيه على الحالة التى يكون فيها الخرج غير محدد. فى هذه الحالة فإن الخرج يعكس حالته.

**Johnson counter**

عداد جونسون، نوع من العدادات الدوارة يتميز بأن له عدد من الحالات ضعف العداد الدوار العادى.

**K****Karnaugh map**

طريقة تخطيطية منظمة لتبسيط المعادلات المنطقية إلى أبسط صورة ممكنة.

**L****Large Scale Integration, LSI**

التكامل على المستوى، درجة من التعقيد فى تصنيع الشرائح الإلكترونية حيث تبلغ كثافة الكونات من 1000 حتى 10000 ترانزستور على الشريحة التكاملية الواحدة.

**Latch**

ماسک، دائرة منطقية ذات خرجين كل منهما عكس الآخر، أنظر flip flop أو القلاب.

**LCD**

شاشات عرض البلور السائلة Liquid Crystal Display، تتميز بالاستهلاك الأقل للفترة ومناسبة للكثير من شاشات العرض.

**Least significant bit, LSB**

البت (الخانة) ذات القيمة الصغرى فى أي رقم وهى البت الموجودة فى أقصى يمين الرقم.

**LED**

الدايدون الضوئى Light Emitting Diode، هو دايدون يشع ضوء من جميع الألوان عند مرور التيار فيه، ويستخدم فى عمل الكثير من شاشات العرض ولمبات البيان.

**Logic**

منطقى. المستوى المنطقى فى الإلكترونيات الرقمية هو تمثيل التعبير الغير حقيقى بتصور والتعبير الحقيقى بواسطه.

**Look ahead**

ينظر للأمام look ahead carry adder المجمع ذو الحمل الأمامى، يتميز بسرعة.

**M****Master slave flip flop**

قلاب مكون من ماسكين، الأول هو السيد master والثانى هو العبد slave. وهذه أحد طرق الحصول على قلاب حساس لأحد حواجز نبضة التزامن.

**Medium Scale Integration, MSI**

التكامل المتوسط، درجة من التعقيد فى تصنيع الشرائح الإلكترونية حيث تبلغ كثافة المكونات من 100 حتى 1000 ترانزستور على الشريحة الواحدة.

**Monostable**

أحادى الاستقرار، دائرة يستقر خرجها على حالة واحدة فقط إما الصفر أو الواحد. إذا تغير الخرج فإن ذلك يكون لفترة محددة ثم يرجع تلقائياً لحالة الاستقرار.

**Most significant bit, MSB**

البت أو الخانة ذات القيمة العظمى وهى البت الموجودة فى أقصى يسار أي رقم ثانى.

**Multiplexer, MUX**

منقى، دائرة إلكترونية تختار واحد من مدخلها وتضعه على الخرج تبعاً لتابع معين.

**Multivibrator**

مذبذب، دائرة يتذبذب خرجها بين الواحد والصفر ولا تستقر على أي واحدة من هذه الحالات.

**N****NAND gate**

بوابة ناند NAND، بوابة آند متبوعة بعاكس، يكون خرجها يساوى صفر في حالة واحدة فقط وهى عندما يكون جميع مداخلها تساوى وحيد.

**Nibble**

٤ بتات. نصف بait.

**Nonvolatile**

غير متطابق، تعبر يطلق على نوع من الذاكرة لا تفقد محتوياتها بانقطاع القدرة مثل ذاكرة القراءة فقط ROM.

**NOR gate**

بوابة نور NOR، بوابة أور متبوعة بعاكس، يكون خرجها واحد في حالة واحدة فقط وهى عندما تكون جميع مداخلها أصفارا.

**NOT gate**

بوابة NOT، بوابة عكس. هي بوابة يكون خرجها عكس دخലها.

**O****Octal**

ثماني، نظام العد الثمانى الذى قاعدته ٨.

**One shot**

أحادي النبضة، أحدى الاستقرار. دائرة عند إثارتها تعطى نبضة واحدة فقط على الخرج.

**Open collector**

يتم أخذ خرج الدائرة المنطقية من خلال ترانزستور مفتوح المجمع. تستخدم هذه الطريقة مع الدوائر ذات الأحمال العالية.

**OR gate**

بوابة أور OR، بوابة "أو". بوابة منطقية يكون خرجها صفر في حالة واحدة فقط وهى عندما تكون كل الدخول أصفارا.

**Oscillator**

مذبذب، مولد إشارة. دائرة تعطى على خرجها موجة متكررة باستخدام نظام تغذية مرتبة في تصميمها.

**Output**

خرج دائرة معينة أو نظام معين.

**Overflow**

فيضان، يحدث في عملية الجمع عندما يزداد عدد بتات الناتج عن عدد بتات أي واحد من العددين المجموعين وبالذات مع الأرقام ذات الإشارة حيث يطغى الحمل من الخانة الأخيرة على خانة الإشارة.

**P****Parallel**

التوازى، ويعنى خروج مجموعة من البيانات على مجموعة من الخطوط فى نفس الوقت.

**Potentiometer**

مقسم جهد ويكون فى العادة من خلال مقاومة متغيرة.

**Power dissipation**

الطاقة المهدورة، وهى حاصل ضرب تيار مصدر القدرة فى جهد مصدر القدرة الذى يغذي أى دائرة أو شريحة إلكترونية.

**Preset**

جعل الخرج يساوى واحد قبل التشغيل بطريقة غير توافقية لا تعتمد على نبضات التزامن.

**Priority encoder**

مشفر مع الأولوية، مشفر يعطى شفرة الدخل ذو الأولوية الأعلى في حالة تنشيط أكثر من دخل في نفس الوقت.

**Product Of Sums, POS**

مضروب المجاميع، طريقة لعرض التعبيرات المنطقية في صورة عملية آند AND على كميات كل منها عبارة عن أور OR لمجموعة متغيرات.

**Propagation**

انتشار، Propagation delay زمن التأخير الناتج عن انتشار الإشارة أو وصول الإشارة من دخل أى دائرة حتى خرجها.

**Pulse**

نبضة، تغير مفاجئ في قيمة الجهد أو التيار من مستوى لآخر ثم إلى نفس المستوى مرة أخرى في زمن صغير.

**Pull up resistor**

مقاومة توصل بين نقطة معينة ومصدر القدرة لضمان أن جهد هذه النقطة سيكون واحد (عالي) عندما تكون غير نشطة. مثل توصيل مقاومة على أى خرج من خلال المجمع مفتوح.

**Pulse Width Modulation, PWM**

تعديل عرض النبضة. هو التحكم في عرض النبضة بالضبط والاتساع حيث ينتج عن ذلك تغير في القيمة المتوسطة للموجة ويستخدم ذلك في التحكم في سرعة أي موتور.

**R****Random Access Memory, RAM**

ذاكرة الاتصال العشوائي، يمكن القراءة أو الكتابة في أي مكان فيها وليس بالضرورة أن يكون

بالتابع. يطلق هذا الاسم بطريق الخطأ على ذاكرة الكتابة والقراءة.

### Read

القراءة، عملية استدعاء البيانات من الذاكرة.

### Register

مجل، دائرة إلكترونية رقمية قادرة على تخزين بيانات وإزاحتها.

### Reset

تصفير، جعل الخرج يساوى صفر. عودة للوضع الأصلي.

### Ring counter

عداد دوار. عبارة عن مسجل إزاحة تم توصيل خرجه من أقصى اليمين كدخل من اليسار ولابد من تسجيل حالة ابتدائية على العداد قبل السماح بدورانها مع نبضات التزامن.

### Ripple

تموجي، Ripple carry adder المجمع ذو الحمل التموجي، Ripple counter العداد التموجي.

### Rise time

زمن الارتفاع، الزمن اللازم لكي تتغير إشارة من ١٠٪ إلى ٩٠٪ من قيمتها.

### Reliable

موثوق به، يمكن الاعتماد عليه، reliability هي معامل الثقة.

### Remainder

الباقي، من عملية القسمة.

### Resistance

مقاومة.

### Reset

إعادة الوضع، يمكن أن تعنى إعادة المعالج أو المتحكم إلى الوضع التلقائي، أو تعنى وضع إشارة معينة تساوى صفر بعد أن كانت واحد.

### Resistance network

شبكة مقاومات. مجموعة من المقاومات داخل غلاف واحد موصولة مع بعضها بطريقة معينة.

### R-S flip flop

قلاب له دخلان R و S محظور فيه أن يكون كل منهما يساوى واحد في نفس الوقت وإلا فإن خواص القلاب تفقد حيث يكون الخرج في هذه الحالة غير مرغوب فيه.

## S

### Sequential circuit

دائرة تتبعية، دائرة منطقية يعتمد خرجها على تتابعات زمنية معينة. تحتوى عناصر ذاكرة. لذلك فالخرج يعتمد على الدخل الحالى والخرج فى لحظات سابقة. تعتمد فى تشغيلها على نبضات

تزامن. من أمثلة هذه النظم العدادات ومسجلات الإزاحة.

### Serial

تتابعى، تتابع بيانات أو نبضات على نفس الخط فى أزمنة متتابعة.

### Serial communication

الاتصالات التتابعية أو المتالية، وهناك العديد من هذه الطرق أو هذه البوتوكولات، ولكن السمة المميزة لهذه الطريقة من التواصل أن الرسالة تخرج فى صورة تتابع من البيانات على مخرج أو سلك واحد.

### Set

جعل الخرج يساوى واحد، وضع الخرج فى حالة معينة، عكس.reset

### Set up time

زمن الاستقرار، وهو الفترة الزمنية التي يجب أن يظل الدخل فيها مستقرا قبل تطبيق الحافة المؤثرة لنبضة التزامن، وإن الخرج لا يتغير لقيمة مستقرة.

### Shift register

مسجل الإزاحة. دائرة منطقية يمكن تسجيل بيانات بها ثم إجراء إزاحة أو دوران على هذه البيانات.

### Sign

الإشارة، وهى إشارة الرقم الذى تكون سالبة أو موجبة.

### Sign bit

خانة الإشارة . فى العادة تكون البت فى أقصى يسار الرقم، تكون واحد إذا كان الرقم سالب، وصفر إذا كان الرقم موجب.

### Small Scale Integration,SSI

التكامل الصغير، درجة من التعقيد فى تصنيع الشرائح الإلكترونية حيث تبلغ كثافة المكونات أقل من ١٠٠ ترانزستور على الشريحة الواحدة.

### Speed Power product

حاصل ضرب السرعة فى الطاقة المهدبة، ويستخدم كمعامل لقياس أداء الشرائح والدواير الإلكترونية الرقمية.

### S-R flip flop

قلاب له دخلان R و S محظور فيه أن يكون كل منها يساوى واحد فى نفس الوقت وإلا فإن خواص القلاب تفقد حيث يكون الخرج فى هذه الحالة غير مرغوب فيه.

### Stack

المكذسة، جزء من الذاكرة RAM توضع فيه البيانات على أساس من يأتي أخيرا يخرج أولا، والمكذسة تكون ضرورية عند التعامل مع البرامج الفرعية أو المقاطعة.

### Stack Pointer

مؤشر المكستة، وهو مسجل يحتوى عنوان آخر مكان تم التخزين فيه في المكستة.

### Stage

مرحلة، مثل مرحلة من مراحل عداد أو مسجل إزاحة وتكون عبارة عن قلاب في هذه الحالة.

### Static Memory, SRAM

ذاكرة استاتيكية، وحدة بناؤها هي القلاب ولا تحتاج لإنشاع بياناتها مثل الذاكرة الديناميكية.

### Strobe

طرف في بعض الدوائر المنطقية، عندما يكون نشط يتغير الخرج تبعاً لحالة الدخل، وعندما يكون غير نشط فإن الخرج لا يرى الدخل.

### Subtractor

طارح، دائرة تقوم بعملية الطرح الثنائي على رقمين مدخلين إليها.

### Sum Of Products, SOP

مجموع المضاريب، طريقة لعرض التعبيرات المنطقية في صورة عملية أور OR على كميات كل منها عبارة عن آند AND لمجموعة متغيرات.

### Synchronous

تواافقى، أو متزامن، أي يتغير بالتوافق مع نبضات تزامن معينة. من أمثلة ذلك العداد التوافقى.

## T

### Terminal Count, TC

العدة الطرفية (النهائية)، الحالة النهائية للعداد. مثل الرقم 9 في العداد العشري التصاعدى.

### Timer

مؤقت، دائرة توقف. يتم تنفيذه إما عن طريق دائرة (وهناك العديد من الشرائح التي تقوم بذلك مثل الشريحة 555)، أو يتم تنفيذه عن طريق برنامج يقوم بعد نبضات تزامن معلومة التردد حتى انقضاء الزمن المحدد كما هو الحال في المحكمات.

### Timing diagram

المخطط الزمني، مخطط بين العلاقة بين مقدار الإشارة والزمن وبالذات حينما يكون هناك أكثر من إشارة ويتم رسماً كلها مع الزمن في نفس المخطط حتى تظهر العلاقة بينها.

### T flip flop

قلاب له دخل واحد اسمه T حيث ينعكس الخرج مع كل نبضة تزامن إذا كان هذا الدخل واحد. وإذا كان هذا الدخل صفر فلا يتغير الخرج.

### Toggle

يعكس، إذا كان الخرج صفر يصبح واحد، وإذا كان واحد يصبح صفر.

### Trailing edge

الحافة الثانية لأى نبضة.

### Trigger

إطلاق، نبضة تعطى لبدء التغيير في قيمة الخرج لدائرة رقمية تبعاً لدخولها.

### Tristate logic

المنطق الثلاثي، دائرة منطقية لها الحالات المنطقية المعروفة لكل الدوائر المنطقية العادية، بالإضافة لحالة ثالثة يكون الخرج فيها مقاومة عالية أو مفتوح.

### Truth table

جدول الحقيقة، يبين الخرج عند جميع الاحتمالات الممكنة للدخل في الدوائر الرقمية.

### TTL, Transistor Transistor Logic

أحد تكنولوجيات تصنيع الشرائح الإلكترونية باستخدام الترانزستور ثنائى القطبية. تتميز بأن جهد الواحد المنطقى 5 فولت والصفر المنطقى هو صفر فولت.

## U

### Ultra large scale Integration, ULSI

التكامل المتزايد، درجة من التعقيد في تصنيع الشرائح الإلكترونية حيث تبلغ كثافة المكونات أكثر من مليون ترانزستور على الشريحة الواحدة.

### Universal gate

بوابة عامة. بوابات تتميز بأنه يمكن بناء نظام إلكترونى كامل باستخدام هذا النوع من البوابات فقط. مثل ذلك بواية الناند NAND وبوابة النور NOR.

### Universal shift register

مسجل إزاحة عام، بخطوط تحكم معينة يمكن الإزاحة من اليمين لليسار أو العكس، ويمكن إدخال البيانات توازى وإخراجها توازى أو العكس، كما يمكن إجراء عمليات الدوران المختلفة.

### Up/down counter

عداد تصاعدى تنازلى، بخط تحكم يمكن جعل العداد بعد تصاعدياً أو تنازلياً.

### USART

الإرسال والاستقبال العام المتزامن وغير المتزامن، وهى فى العادة شريحة يمكن من خلالها إرسال واستقبال البيانات بالطريقة المتزامنة والطريقة غير المتزامنة وهى تكون أحد المكونات الأساسية فى الكثير من المحكمات.

## V

### Very Large-Scale Integration, VLSI

التكامل العالى جداً، درجة من التعقيد فى تصنيع الشرائح الإلكترونية حيث تبلغ كثافة المكونات من ١٠٠٠٠٠ حتى مليون ترانزistor على الشريحة الواحدة.

### Volatile

متطابير، تعبر يطلق على الذاكرة التى تفقد محتوياتها بانقطاع مصدر القدرة. مثل ذاكرة القراءة والكتابة .RAM

## X

### XOR gate

بواية إكس أور، تعطى واحد في حالة اختلاف الدخلين وصفر في حالة تساويهما.

### XNOR gate

بواية إكس نور، عكس البوابة إكس أور.

## W

### Word

كلمة، ١٦ بت، أو ٢ بايت. وحدة من وحدات تخزين البيانات الرقمية.

### Write

الكتابة، عملية تخزين البيانات في الذاكرة.

## Z

### Z 80

معالج ٨ بت، وهو أحد إصدارات شركة zilog وقد نزل السوق في عام ١٩٧٥ تقريباً. وهو معالج سهل البرمجة يمكن استخدامه في الكثير من أغراض التحكم، وإن طغت عليه المتحكمات ٨ بت هذه الأيام.