

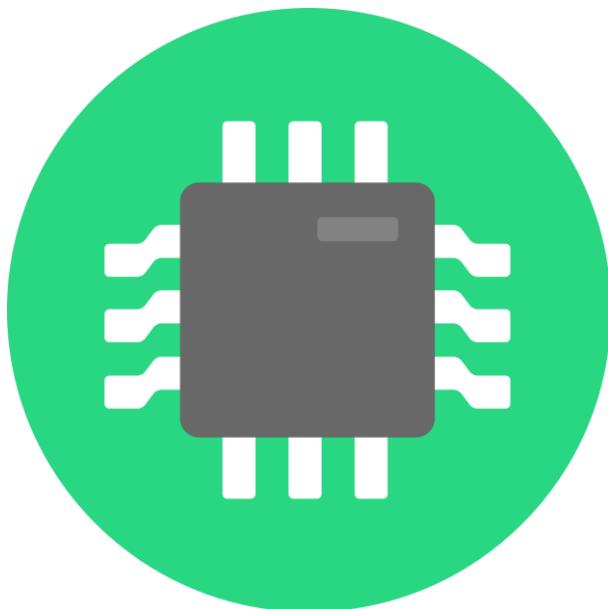


رخصة الكتاب



Simply AVR

From Blinking led to RTOS *“Final Beta Edition”*





رخصة الكتاب

"تعلم AVR ببساطة" بصيغة PDF منشورة مجاناً للجميع تحت رخصة الإبداع Creative Common v4 CC-NC-SA بشروط: عدم الاستغلال التجاري.



النسخة الإلكترونية من كتاب المشاعي الإصدارة الرابعة النسبة - المشاركة بالمثل -

رخصة المشاع الإبداعي-CC-NC (غير تجارية) لك كامل الحق في نسخ وتوزيع وتعديل أو الإضافة أو حتى طباعة الكتاب ورقياً كما تشاء (أو أشجعك على ذلك أيضاً) شرط عدم استغلال الكتاب تجاريًّا بأي صورة مباشرة أو غير مباشرة، يجوز طباعة وتوزيع الكتاب شرط أن يبيع بتكلفة الطباعة فقط.

المشاركة بالمثل-SA إذا تم اشتقاق أي عمل من هذا الكتاب بصورة إلكترونية أو مادية مثل عمل كتاب آخر أو محاضرة تعليمية (أو حتى كورس متكامل) أو فيديو فيجب أن يتم بصورة مجانية و بنفس الرخصة (المشاع الإبداعي: النسبة، المشاركة بالمثل، الغير تجارية). يمكنك التعرف أكثر على رخصة المشاع الإبداعي من الموقع الرسمي creativecommons.org

جميع الكتب الإلكترونية من سلسلة "تعلم ببساطة" مجانية و منشورة بنفس الرخصة

يمكنك تحميل السلسلة من الموقع

http://simplyarduino.com/?page_id=889

جميع العلامات التجارية المذكورة في الكتاب ملك لأصحابها

عبد الله علي عبد الله

صفحة جديدة



إهادء

أبي و أمي ... لولاكم ما تعلمت حرفاً ..

أساتذة هندسة الحاسوبات بجامعة حلوان وأخص منهم،

أ.د. محمد العدوسي

شكراً لأنك أحبيبـتـ العربيةـ،ـ شـكـراـ جـزـيـاـ عـلـىـ كـلـ كـتـبـكـ الـرـائـعـةـ،ـ فـهـيـ ذـخـرـ لـلـأـمـةـ

أ.د. علاء حمدي

شكراً على تبسيطك للعلم، يعلم الله كم البهجة الذي أدخلته على قلوب الطلبة ())

د. أحمد يوسف

شكراً على إخلاصك، تحملـكـ لـلـطـلـبـةـ وـالـإـصـرـارـ عـلـىـ إـيـصـالـ الـعـلـمـ،ـ شـكـراـ



شكراً لكل من شارك

م. أحمد أسامة - شكرأً للمساهمة بفصل الـ **UART**
م. يحيى طويل - شكرأً على نصائحك ومقالات "عاديات" الرائعة

شكراً لكل من ساهم بمراجعة الكتاب

م. حمدي سلطان، م. سعيد الشايب، أحمد م. أبو زيد، إسلام الليثي، محمد عويس،
هاجر شرف، لميس الموصلي.

ولكل من ساهم بنصيحة أو تعليق بناء، شكرأً لكم جميعاً



الفهرس

1. حول الكتاب - الإصدار 1.0	
11. فصول الكتاب.....	
15. لماذا سنستخدم C – ANSI ؟.....	
16. حرب المُتحكمات - من هو الأفضل الـ AVR أم الـ PIC ؟.....	
2. مقدمة عن الأنظمة المدمجة	
23. 1.1 معنى النظام المدمج..... Embedded System	
24. 1.2 مكونات النظام المدمج.....	
25. 1.3 مراحل تطوير الأنظمة المدمجة.....	
3. نظرة عامة على مُتحكمات AVR	
35. 2.1 تركيب المُتحكم الدقيق وعمارية AVR..... AVR	
37. 2.2 مميزات عمارية الـ AVR..... AVR	
39. 2.3 كيف تختار بين عائلات الـ AVR المختلفة..... AVR	
42. 2.4 قراءة دليل البيانات Datasheet..... Datasheet	
43. 2.5 الخصائص العامة للمُتحكم ATmega16/ATmega32..... ATmega16/ATmega32	
47. 2.6 أطراف المُتحكم ATmega16..... ATmega16	
49. 2.7 عائلة ATTiny..... ATTiny	
52. 2.8 تمارين إضافية.....	
53. 2.9 مراجع إضافية.....	
4. تجهيز أدوات التجارب	
56. 3.1 المُبرمجات.....	
62. 3.2 المكونات الإلكترونية.....	
65. 3.3 أدوات إضافية.....	
66. 3.4 تجهيز البرمجيات.....	
73. 3.5 مراجع إضافية.....	
5. أساسيات التحكم	
77. 4.1 المثال الأول: Hello World..... Hello World	
89. 4.2 شرح المثال الأول وأساسيات برمجة الـ AVR..... AVR	
96. 4.3 المثال الثاني: استخدام 4 دايويد ضوئي.....	
99. 4.4 المثال الثالث: تشغيل جميع أطراف PortA, Port B..... PortA, Port B	
102. 4.5 المثال الرابع: تشغيل المقاطعة السباعية segment7..... segment7	
107. 4.6 المثال الخامس: قراءة الدخل الرقمي Inputs reading..... Inputs reading	
110. 4.7 خاصية الـ Pull Up & Pull Down Resistor..... Pull Up & Pull Down Resistor	
113. 4.8 خاصية الـ Internal Pull-Up..... Internal Pull-Up	
114. 4.9 المثال السادس: تشغيل 3 دايويدات + 3 مفاتيح.....	
117. 4.10 بيمبونج effect & De-bouncing..... Bouncing effect & De-bouncing	
119. 4.11 حساب المقاومة المستخدمة قبل الأحمال.....	



121	توصيل أحمال بتيارات كبيرة
123	تشغيل المحركات DC
125	4.12 تشغيل المحرك في كلا الاتجاهين
129	5. قواعد لغة السي للأنظمة المدمجة
130	5.1 أنواع البيانات في الأنظمة المدمجة Data-types
135	5.2 العمليات الحسابية Arithmetic Operations
136	5.3 العمليات المنطقية Logic Operation
139	5.4 عمليات الإزاحة Shift operations
142	5.5 التحكم على مستوى البت الواحد Single Bit
144	5.6 القراءة من بت واحدة Read single bit
146	6. الفيوزات، الحماية، الطاقة وسرعة التشغيل
147	6.1 Fuses & Lockbits
154	6.2 LockBits
155	6.3 المذبذبات والـ Clock Source
162	6.4 قيم الفيوزات لضبط السرعة
166	6.5 الطاقة وسرعة تشغيل المُتحكمات
169	6.6 كيف تبرمج الفيوزات
171	6.7 كيف تعالج الفيوزات المُبرمجة بصورة خاطئة؟
174	7. المُقاطعة Interrupt
175	7.1 مقدمة عن المُقاطعة The interrupt
177	7.2 المثال الأول: تشغيل المُقاطعة INTO
185	7.3 المثال الثاني: تشغيل المُقاطعة INT1 مع INT0
188	8. الاتصال التسلسلي بروتوكول UART
189	8.1 مقدمة عن الاتصال التسلسلي
192	8.2 التسلسلي الغير متزامن Asynchronous
194	8.3 تهيئة الـ UART الداخلي لمُتحكمات AVR
196	8.4 المثال الأول: تهيئة الـ UART للعمل كمرسل
200	8.5 المثال الثاني: تهيئة الـ UART للعمل كمستقبل
202	8.6 المثال الثالث: الإرسال والاستقبال في وقت واحد
205	8.7 إرسال مجموعة بيانات مثل السلاسل النصية
209	8.8 دوال إضافية
212	9. المُحول التناضري-الرقمي ADC
213	9.1 مقدمة عن المُحول التناضري-الرقمي ADC
215	9.2 تركيب الـ ADC داخل المُتحكم ATmega16
217	9.3 المثال الأول: قراءة جهد متغير باستخدام مقاومة متغيرة
224	9.4 حسابات الـ ADC
227	10. المعالج التمهيدي وصناعة المكتبات البرمجية
228	10.1 الأوامر التنفيذية والأوامر التوجيهية
228	بعض استخدامات C - preprocessor



229	قواعد الأوامر التوجيهية C - preprocessor syntax
232	function-like macros
232	قواعد كتابة الماكرو macros syntax
233	مراجع إضافية
234	تصميم المكتبات البرمجية في لغة السي
235	خطوات صناعة المكتبة
238	تجربة المكتبة في برنامج ATmel studio
245	11. أنظمة الوقت الحقيقي RTOS
246	مقدمة عن أنظمة الوقت الحقيقي Real Time Systems
247	طرق تصميم الأ- Real Time Embedded systems
250	كيف تعمل النواة RTOS Kernel
251	مقدمة عن نظام FreeRTOS
252	الهيكل البرمجي للRTOS
253	تشغيل FreeRTOS على جميع مُتحكمات AVR
265	المثال الأول: Blinking 3 leds with 3 tasks
271	12. الملحقات الإضافية
272	ملحق: تنصيب برنامج CodeBlocks على نظام ويندوز
278	ملحق: ترجمة الملفات باستخدام makefile
282	ملحق: رفع ملف الأ- Hex على المُتحكم الدقيق.
287	ملحق: كيف تستخدم لوحات آردوينو لتعلم برمجة AVR
291	قائمة المراجع





حول الكتاب - الإصدار 1.0

هذا الكتاب موجه إلى كل من يرغب بدخول مجال تطوير النظم المدمجة Embedded Systems بصورة احترافية والبدء بتعلم أساسيات هذا المجال الممتع بأسلوب عملٍ معتمد على التجارب.

لقد حرصت على أن يكون الشرح باللغة العربية بخطوات يسيرة ومفصلة ومع ذلك سأحافظ على استخدام بعض المصطلحات الإنجليزية في الشرح حتى تتعاد على هذه المصطلحات ويصبح من السهل لك قراءة المراجع الإنجليزية.

الكتاب ليس مصمم ليكون مرجع شامل بقدر ما هو موجه ليكون بداية انطلاق نحو احتراف المجال، الحقيقة أن هذا العلم لا يمكن احتواه أبداً في كتاب أو مرجع حتى وإن كان 1000 صفحة، لذا قمت بإضافة مصادر تعليمية بعد كل فصل تشرح المزيد من التجارب والمعلومات عن نقاط هذا الفصل فاحرص على قراءة هذه المصادر الإضافية لستزيد من العلم.

فصول الكتاب

الفصل الأول: مقدمة سريعة عن الأنظمة المدمجة والمكونات المستخدمة في بنائها وكيفية اختيار هذه المكونات لتحقيق أقصى استفادة بأقل سعر وشرح عام لمراحل التطوير بداية من الفكرة وإنتهاءً بالمنتج الذي يباع للمستهلك.

الفصل الثاني: يقدم شرح مبسط للتركيب الداخلي للمتحكم الدقيق مع شرح لخواص ومميزات المتحكمات من نوع AVR وكيفية قراءة دليل البيانات Datasheet الخاصة بها واستخلاص أهم المعلومات.

الفصل الثالث: يوضح هذا الفصل الأدوات التي ستسخدمها في تطوير الأنظمة المدمجة سواء كانت العتاد "المكونات الإلكترونية" Hardware أو الأدوات البرمجية (Softwares) ToolChain.

الفصل الرابع: من هنا نبدأ رحلة تعلم المتحكمات الدقيقة وسنبدأ مع أساسيات تشغيل أطراف المتحكم الدقيق وتشغيل المنافذ لعمل كدخل أو كخرج GPIO. كما سنقوم بمجموعة من التجارب لتشغيل العناصر الإلكترونية البسيطة مثل LEDs, Switchs, 7-Segments..الخ.

الفصل الخامس: شرح لأهم القواعد والصيغ الشهير للغة السي المعيارية المستخدمة بشكل كبير في تطوير الأنظمة المدمجة. تتميز الصيغ المعيارية بإمكانية تطبيقها على مختلف المتحكمات الدقيقة طالما أن المترجم الخاص بها يدعم لغة السي.

الفصل السادس: شرح للإعدادات المتقدمة لمتحكمات AVR مثل مفهوم الفيوزات ووظائفها المختلفة مثل تغيير سرعة التشغيل Clock Rate واستهلاك الطاقة، حماية البرامج الموجودة على المتحكم من السرقة أو التعديل وتشغيل بعض الخصائص المتقدمة الأخرى.

الفصل السابع: سنتعرف في هذا الفصل على كيفية تشغيل المقاطعات الخارجية External Interrupts وفائدة هذه الخاصية الرائعة التي تتيح صناعة تطبيقات ذات استجابة عالية السرعة للأحداث الخارجية.

الفصل الثامن: شرح أحد أشهر طرق إرسال البيانات بصورة تسلسلية بين المتحكمات الدقيقة والعالم الخارجي وذلك عبر بروتوكول UART والذي يعتبر أشهر بروتوكول معياري لتبادل البيانات.

الفصل التاسع: في هذا الفصل سنتعرف على كيفية قراءة الجهود الكهربائية المتغيرة Analog وتحويلها إلى قيم رقمية وذلك باستخدام المحوّل التناهري - الرقمي المدمج داخل متحكمات AVR. حيث يمكن استغلال هذا المحوّل في قراءة الحساسات التناهيرية أو أي عنصر إلكتروني له خرج كهربائي متغير.

الفصل العاشر: شرح أكواد C preprocessor حيث سنتعرف على الفارق بين الأوامر التنفيذية والأوامر التوجيهية وأهميتها بصورة مفصلة مثل الأمر #define وكذلك سنتعرف على كيفية صناعة المكتبات البرمجية libraries. مع شرح مثال لعمل uart driver على صورة مكتبة.



الفصل الحادي عشر: طرق استخدام أنظمة تشغيل الوقت الحقيقي Real Time OS لتشغيل المهام المتعددة Multitasking وأنظمة الاستجابة السريعة. حيث سيتم تناول نظام FreeRTOS في هذا الفصل باعتباره أفضل نظامRTOS مجاني (ومفتوح المصدر).

وفي النهاية مضاف مجموعة من الملاحق التدريبية، كل ملحق يشرح مهارة تقنية مختلفة

الكتاب موجه خصيصاً إلى طلبة الكليات الهندسية مثل:

- ✓ تخصص هندسة الحاسوبات والاتصالات.
- ✓ تخصص هندسة الإلكترونيات والكهرباء.
- ✓ تخصص ميكترونكس.
- ✓ هواة الإلكترونيات بشكل عام.

يتطلب قراءة هذا الكتاب بعض المعرفة المُيسقة:

- ✓ أساسيات لغة السيC بشكل عام مثل استخدام المتغيرات والثوابت if - for - while
 - ✓ أساسيات الإلكترونيات والكهرباء مثل المقاومات، المكثفات، البطاريات .. إلخ
- إذا لم يكن لديك أي خبرة بما سبق فأنصحك بقراءة المراجع التعليمية العربية (في نهاية الكتاب) حيث تحتوي على موارد عربية رائعة لشرح علم الإلكترونيات من الصفر

لتشغيل البرمجيات التي سنتخدمها في تطبيق الأمثلة المذكورة في الكتاب ستحتاج أن تمتلك حاسب آلي بالإمكانيات التالية على الأقل:

- معالج بنديوم 4 أو أعلى مثل i7 - i5 - Core i3 - Core2Due
- ذاكرة عشوائية RAM سعة 1 جيجا أو أكثر
- مساحة تخزينية فارغة 5 جيجا على الأقل
- نظام تشغيل Windows أو Linux (مع العلم أن التطبيق الأساسي سيكون على نظام ويندوز).

يمكنك الحصول على نسخة مجانية (مدى الحياة) من برنامج Atmel studio من الموقع الرسمي مع العلم أن الموقع يتطلب تسجيل حساب (مجاني) لتحميل البرنامج.

<http://www.atmel.com/tools/ATMELSTUDIO.aspx>

كما يمكنك الحصول على نسخة مجانية (لمدة شهر) من برنامج المحاكاة بروتس Protues من الموقع الرسمي <http://www.labcenter.com>

سيتم استخدام كلا البرنامجين بصورة أساسية في شرح التجارب المذكورة في الكتاب.

ملاحظة: برامج المحاكاة مثل بروتس تتطلب استهلاك قدر كبير من الذاكرة و المعالج لذا احرص على إغلاق أي تطبيقات أخرى لا تستخدمها عند تشغيل برنامج بروتس

سيتركز الكتاب على شرح المُتحكمات الدقيقة من نوع AVR - 8 bit المصممة بواسطة الشركة العملاقة ATmel لما لها من مميزات رائعة، وسيكون الشرح مبني على لغة السيC المعيارية أو كما تعرف باسم ANSI - C (C89, C99) وسيتم استخدام المترجم AVR-GCC المضمن مع برنامج ATmel Studio وللنظام Arduino IDE, Code vision, MikroC .



أتخيل أنك بعد هذه المقدمة قد تتساءل .. لماذا سيرتكز الشرح على لغة السي المعيارية C – ANSI و سبب اختياري للـ AVR بدلاً من الـ PIC ...

لماذا سنستخدم C – ANSI ؟

لماذا نستخدم C – ANSI بدلاً من اللغات ومعايير البرمجة الأخرى مثل Flow Code أو Bascom بالرغم أن هذه الطرق قد تكون أسهل في البرمجة؟ لكنفهم الإجابة علينا أولاً أن نتعرف على كلمة ANSI وهي اختصار المعهد الوطني الأمريكي للمعايير (ANSI)

هذا المعهد قام بوضع معيار موحد للغة السي وذلك حتى تصبح الأكواد المكتوبة بها صالحة على منصات مختلفة (حتى وإن طلبت تعديلات بسيطة). فمثلاً يمكنك كتابة برنامج بلغة السي على نظام ويندوز ومن ثم تقوم بعمل ترجمة له دون تعديل ليعمل على نظام لينكس أو العكس وباستخدام نفس المترجم Compiler.

إن تعلم لغة السي المعيارية والتدريب على تقنيات كتابة الأكواد بها يعطيك القدرة على التعامل مع أنواع كثيرة جداً من المُتحكمات الدقيقة، فمثلاً بعد انتهاءك من هذا الكتاب ستكون قادرًا على قراءة الأكواد المكتوبة لمتحكمات ARM بدون مجهود كبير، بل قد تجد أن الأوامر شبه متطابقة في الكثير من الحالات (باختلاف أسماء المُسجّلات Registers التي ستحدث عنها في الفصول القادمة). ليس هذا فحسب بل ستجد أن تعلم برمجة أي مُتحكم آخر أصبحت عملية سهلة جداً طالما أن المترجم الخاص بهذا المُتحكم يدعم السي المعيارية.

من أجل هذه الأسباب سنتعامل فقط مع البرامج التي تدعم هذه اللغة مباشرة مثل CodeVision و CodeBlocks و Atmel Studio أما باقي البرامج مثل تجعلك تتعلم بعض الممارسات السيئة في كتابة الكود والتي قد لا تتوافق مع معايير الـ C – ANSI وبعض البرامج تضيف مكتبات تغير طريقة البرمجة بالكامل (مثل آردوينو) وهذا أيضاً لا أنصحك باستعماله إذا كنت تهدف لاحتراف تصميم الأنظمة المدمجة.



حرب المُتحكمات - من هو الأفضل الـ AVR أم الـ PIC ؟



US.

Atmel

هذا السؤال دائماً ما يتبارد لكل من يعمل أو بدأ يدخل مجال الأنظمة المدمجة، دائماً سنجد هذا الصراع القائم بين فريق متحمس للـ AVR وآخر للـ PIC، الحقيقة أن حسم هذا الصراع أمر صعب للغاية لكن اسمح لي أن أعرفك على بعض جوانب هذه الحرب ..

في البداية لنعرف بشيء هام، في مجال النظم المدمجة لا يوجد ما يسمى "ما هو أفضل مُتحكم دقيق" بصورة مطلقة ولكن هناك "من الأنسب" للاستخدام في تطبيق معين

في بعض الأحيان نحتاج أن نصمم نظام تحكم بسعر رخيص جداً ولا نحتاج لقدرات خارقة أو مُتحكمات متطرفة لتشغيله لذا نبحث عن المُتحكم "الأرخص" والذي يكفي فقط لهذه المهمة لذا لا تستغرب أن علمت أن المُتحكمات (8-bit) STM8 تعتبر من أكثر المُتحكمات مبيعاً في العالم لأنها أرخص من كل من الـ AVR والـ PIC بـ 8 بت وتتفوق عليهم في تقديم قدرات مناسبة بسعر منخفض.

لكن دعنا نعود للسؤال الأصلي والمتبقي في حرب طويلة بين المطوريين .. من الأفضل الـ AVR أم PIC ؟ للإجابة سأقوم بعقد بعض المقارنات التقنية والمالية بين مُتحكمات كل من AVR - 8 bit و الـ PIC - 8 bit

أولاً : مقارنة السرعة

هنا سنجد أن مُتحكمات الـ AVR - 8 bit تتفوق بفارق كبير جداً ويعتبر أدانها أسرع بنحو 4 أضعاف من مثيلتها في الـ PIC - 8 bit وذلك لأن مُتحكمات AVR تستطيع أن تنفذ عدد أوامر في الثانية الواحدة = التردد الذي تعمل به أما الـ PIC فيمكنه تنفيذ رُبع هذا العدد

مثلاً لو معنا مُتحكم AVR و PIC وكلاهما يعمل بتردد = 16 ميجاهرتز (16 مليون هرتز) سنجد أن الـ AVR يمكنه تنفيذ 16 مليون أمر برمجي في الثانية الواحدة بينما الـ PIC بنفس السرعة يستطيع أن ينفذ فقط 4 مليون أمر في الثانية الواحدة.

يرجع هذا الأمر إلى تقنية الـ Pipeline التي تتميز بها جميع مُتحكمات AVR ولا تتوارد إلا في بعض فئات الـ PIC المتطرفة نسبياً.

أيضاً تحتوي معظم شرائح AVR على بعض الأدوات التي تسرع من تنفيذ الأوامر مثل الـ Hardware multiplier وهي وحدة معالجة لعمليات الضرب الحسابية يمكنها تنفيذ عملية الضرب في 2 نبضة فقط (ستتعرف على النبضات ومفهوم التردد في فصل الفيوزات والتحكم في سرعة التشغيل). بينما مُتحكمات PIC المماثلة لا تحتوي على هذا الأمر وقد تستغرق نفس عملية الضرب عليها نحو 40 ضعف الوقت المطلوب على الـ AVR.

ثانياً: التصميم الداخلي ومعالجة البيانات

عندما نكتب برنامج بلغة التجميع Assembly نجد فارقاً ضخماً بين كليهما حيث يتمتع الـ AVR بوجود 32 مُسجل عام Register "ريجستر" يمكن استخدامه في معالجة وتخزين البيانات المؤقتة بسرعة وكفاءة بينما يجرك الـ PIC على استخدام مُسجل واحد فقط (مُسجل التراكم Accumulator) في معظم الأوامر وهذا يعني أن البرامج المكتوبة على الـ AVR أكثر كفاءة وأسرع بكثير من البرامج المكتوبة على الـ PIC.

مثال على ذلك، البرنامج التالي مكتوب بلغة السي ومصمم لكي يبحث عن أكبر قيمة داخل مصفوفة من الأرقام Array وتم تشغيل نفس الكود على مجموعة من المُتحكمات الدقيقة مثل PIC18F و ATmega16 و MSP.

```
int max(int *array)
{
    char a;
    int maximum=-32768;
```



```

for (a=0;a<16;a++)
    if (array[a]>maximum)
        maximum=array[a];
return (maximum);
}

```

الجدول التالي يوضح عدد циклов (cycles) لتنفيذ التحديد النهائى (بالميكروثانية) للكود السابق على مختلف المتحكمات، لاحظ كيف أن atmega16 يعمل بسرعة 16 ميجا إلا أنه استطاع أن يتفوق على كل من PIC16C74 و ذلك الذي يعمل بسرعة 40 ميجاهرتز.

المقارنة
ستجدها داخل

ملاحظة:
ال الكاملة
ملف

Device	Max Speed [MHz]	Code Size [Bytes]	Cycles	Execution Time [uS]
ATmega16	16	32	227	14.2
MSP430	8	34	246	30.8
T89C51RD2	20	57	4200	210.0
PIC18F452	40	92	716	17.9
PIC16C74	20	87	2492	124.6
68HC11	12	59	1238	103.2

المرفق مع الكتاب وهو ملف رسمي من شركة Atmel يوضح مميزات هذه العائلة من المتحكمات الدقيقة AVR_introduction

ثالثاً: استهلاك الطاقة

هنا يتفوق الـ PIC على AVR بفارق واضح، حيث تميز متحكمات الـ PIC باستهلاك منخفض للطاقة (سواء على مستوى فارق الجهد أو التيار الكهربائي). ومع ذلك نجد شركة Atmel قد حسنت كثيراً بعض إصدارات الـ AVR بتقنيات استهلاك منخفضة للطاقة مثل Pico Power Save لكن ستظل متحكمات الـ PIC أفضل من الـ AVR في هذا الجانب.

رابعاً: البرمجة والدعم المجتمعي

شركة Atmel منذ بداية تصنيع الـ AVR قد اعتمدت على مترجمات compilers مفتوحة المصدر وتدعيم الـ C – ANSI مباشرة مثل AVR-GCC المجاني، مما تسبب في جعلها الخيار المفضل لدى الهواة والمحترفين (وهو نفس السبب الذي جعل مصممي لوحة Arduino يختارون شرائح الـ AVR بدلاً من الـ PIC لصناعة Arduino). أما شركة Microchip فقد اتخذت مساراً مختلفاً، حيث نجد أن برنامج MPLAB يخالف الـ C – ANSI خاصة عند كتابة برامج لعائلات مثل PIC16F مما يجعل تعديل الأكواد المكتوبة بها لاستخدامها مرة أخرى أو نقلها لمتحكمات أخرى عملية صعبة.

هنا مجدداً يتفوق الـ AVR، كما أنه هناك دليل واضح أيضاً على التفوق القوي وهو مدى كبر حجم "مجتمع" الهواة والمطوريين والمواقع الإلكترونية الأجنبية التي تدعم الـ AVR والتي لن تجد مثيلها في حالة الـ PIC.

خامساً: السعر مقارنة بالمميزات المدمجة

في الأسواق المحلية تعتبر متحكمات الـ AVR والـ PIC متقاربة جداً في السعر لنفس العائلات (عائلة المتحكمات: هي مجموعة من المتحكمات الدقيقة التي



تشترك في خصائص وامكانيات مشتركة مثل سعة الذاكرة أو الحجم أو الطاقة المستهلكة .. الخ) فمثلاً نجد في السوق المصري أن سعر الـ ATmega16 مساوي تقريباً للـ PIC16F877a (25 جنية مصرى وقت كتابة هذه السطور وهو ما يساوي 3.5 دولار)

لكن نجد أن ATmega16 يوفر قدرات مضاعفة مقارنة بسعر Pic16F منها مثلاً: الأتميجا أسرع 4 مرات من البيك + توفير نحو 3 أضعاف عدد مخارج الـ PWM وهو ضعف معدل سحب التيار لكل طرف من أطراف المُتحكم كما أن الذاكرة في الـ ATmega16 تساوي مرة ونصف حجم الذاكرة في الـ PIC16F877.

يجب التنوية أن هذه الأسعار هي أسعار محلية وقد تختلف من دولة لأخرى أو عند الشراء بكميات كبيرة

نستنتج من ذلك أنه في حالة الرغبة بتطوير نظام سريع الاستجابة أو يقوم بعمليات حسابية معقدة وبسعر مناسب فإن الـ AVR هو الخيار الأمثل لأن الأنظمة المدمجة المعتمدة على المُتحكمات 8 بت الرخيصة



سادساً: التوافر الكمي في الأسواق

هنا نجد أن شركة MicroChip (المصنعة لـ PIC) تتغنى على ATmel فكلا السوقين المحلي والعالمي نجد أن منتجات Microchip متوفرة ويسهل الوصول إليها مقارنة بـ AVR .

هذه هي أهم الأسباب التي قد تجعلك تفضلـ AVR عنـ AVRـ وـ قد تحسـ الصـراعـ بـيـنـ المـتـحـكـمـاتـ الـ 8ـ بـتـ،ـ لـكـنـ مـجـدـدـاـ تـذـكـرـ أـنـهـ فـيـ بـعـضـ الـحـالـاتـ يـكـونـ عـلـيـكـ اـخـتـيـارـ مـتـحـكـمـ لـأـنـهـ الـأـنـسـبـ وـ الـأـفـضـلـ سـعـراـ.

وـتـسـتـمـرـ الـحـربـ مـعـ مـقـارـنـاتـ إـضـافـيـةـ

إـذـاـ أـحـبـيـتـ أـنـ تـقـرـأـ الـمـزـيدـ عـنـ حـرـبـ الـمـقـارـنـاتـ بـيـنـ الـAVRـ وـ الـPICـ فـعـلـيـكـ بـهـذـهـ الـمـقـارـنـاتـ الـرـائـعـةـ وـ الـتـيـ سـتـوـضـحـ جـوـانـبـ إـضـافـيـةـ مـنـ هـذـهـ الـمـقـارـنـاتـ

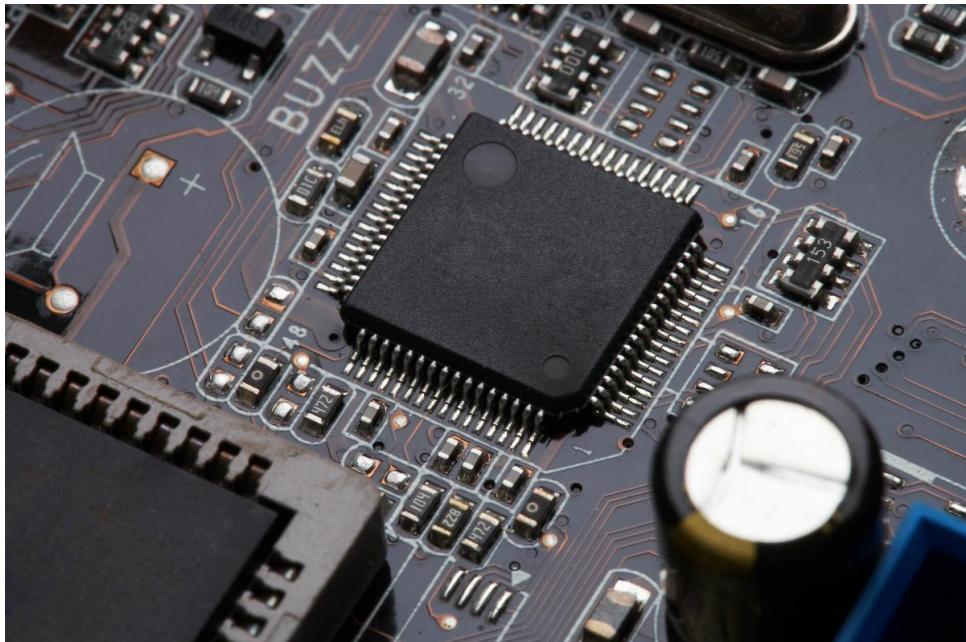
- www.kanda.com/pic-vs-avr.php
- www.youtube.com/watch?v=DBftApUQ8QI
- arstechnica.com/civis/viewtopic.php?f=11&t=409115
- stackoverflow.com/questions/140049/avr-or-pic-to-start-programming-microcontroller



.....صفحة جديدة



1. مقدمة عن الأنظمة المدمجة



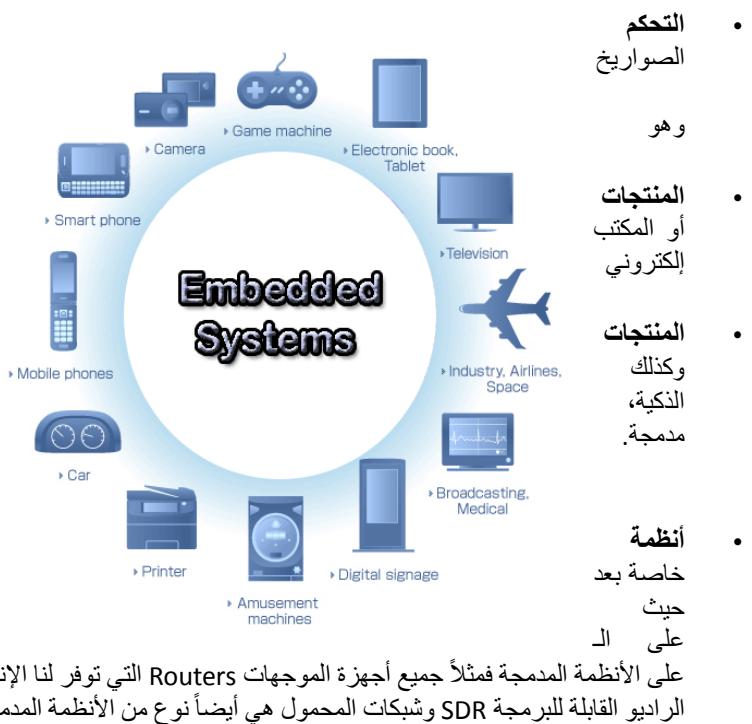
- ✓ معنى النظام المدمج
- ✓ مكونات الأنظمة المدمجة
- ✓ مراحل تطوير المنتجات الإلكترونية المعتمدة على الأنظمة المدمجة
- ✓ كيفية اختيار المُتحكم الدقيق المناسب



1.1 معنى النظام المدمج Embedded System

النظام المدمج أو كما يسمى في بعض الأحيان "النظام المضمن" هو أي نظام حاسوبي صغير الحجم يقوم بمجموعة من الوظائف التي تخدم أداة أو منتج معين، وغالباً لا تباع هذه الأنظمة المدمجة للناس مباشرة ولكنها تكون "مدمجة" مع منتج معين، فمثلاً عند شراء سيارة حديثة أو فرن ميكرويف أو غسالة كهربائية أو حتى مكيف هواء فإنك ستجد أن جميع هذه المنتجات أصبحت تحتوي على حواسيب صغيرة تقدم وظائف تحكم ذكية مما يجعل كل المنتجات السابقة تحتوي على نظم مدمجة.

تستخدم الأنظمة المدمجة في مجموعة واسعة جداً من التطبيقات، أشهرها:



الآلي مثل الأنظمة المدمجة الموجودة في المصانع، الطائرات، والأقمار الصناعية وأي ماكينة تعمل بصورة تلقائية (أوتوماتيكية) هذه الأنظمة جميعها تصنّم لغرض واحد فقط التحكم في منتج معين.

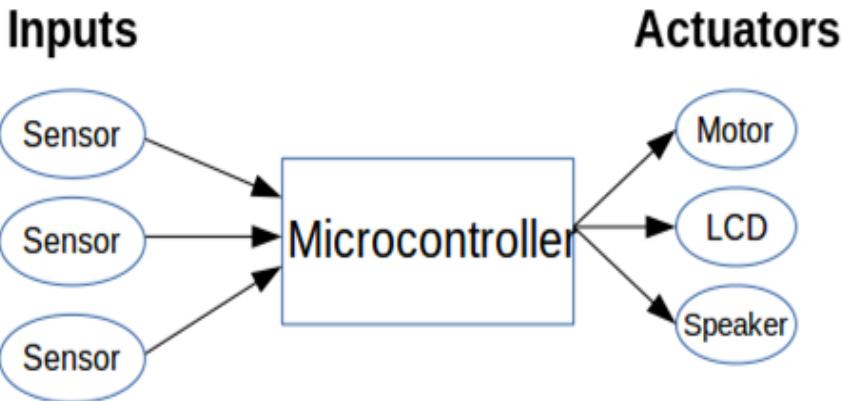
الخدمية مثل المنتجات التي عادة نشتريها لأنفسنا في المنزل مثل مكيف الهواء أو الميكرويف الذي يحتوي على نظام تحكم في الحرارة.

الترفيهية مثل منصات الألعاب Xbox, Gameboy, Wii المنتجات التي أصبحت تحمل وصف "ذكية" مثل الهواتف الساعات الذكية وحتى أنظمة التلفاز الحديثة جميعها تعتبر أنظمة

الاتصالات الحديثة والتي لها نصيب كبير من هذا المجال ظهور تقنيات الاتصال اللاسلكي مثل Wifi وـ Bluetooth تحولت الأجيال القيمة من أنظمة الاتصالات التي كانت تعتمد على تقنيات المعالجة الرقمية المعتمدة Analog Electronics إلى تقنيات المعالجة الرقمية المعمدة على الأنظمة المدمجة.

1.2 مكونات النظام المدمج

- عادة تكون النظم المدمجة من 3 مكونات رئيسية المُتحكم الدقيق MicroController والذي يعتبر العقل المُتحكم في النظام.
 - أدوات الإدخال Input devices مثل الحساسات المختلفة، أزرار الضغط أو أي وسيلة إدخال معلومات للمُتحكم.
 - أدوات إخراج Output devices والتي تُسمى في بعض الحالات Actuators وتعتبر كل ما يتحكم به الـ Microcontroller مثل المحركات Motors، الشاشات LCD، سماعات صوتية ... الخ.
- يتم اختصار أدوات الإدخال والإخراج بكلمة I/O وهي اختصار (Input/Output Devices)



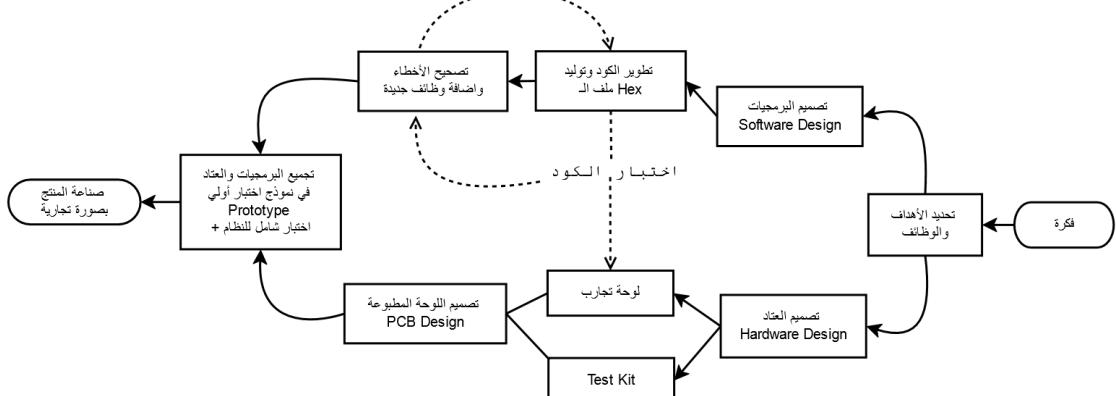
1.3 مراحل تطوير الأنظمة المدمجة

قبل أن نبدأ تعلم صناعة الأنظمة المدمجة علينا أن نفهم الخطوات التي تساعدك على التخطيط لمشروع ناجح وفعال، فمثلاً إذا جاءتك فكرة لجهاز رائع، كيف تنفذها؟ ما هي الأدوات التي ستستخدمها؟ ما هي مراحل تطوير المشروع لتصل إلى منتج نهائي؟..

الصورة التالية توضح الخطوات التي يتبعها مصممو الأنظمة المدمجة في تطوير أي منتج بداية من الفكرة حتى صناعة المنتج بصورة تجارية، كما نرى هناك مسارات أساسية وهما تصميم **hardware** وتصميم **software**.

أولاً: مراحل تطوير برماج المُتحكمات الدقيقة

مثل جميع أنظمة الحواسيب في العالم نجد أن المُتحكمات الدقيقة لا يمكنها أن تعمل دون برنامج يكتب بداخلها وهذا



البرنامج يجب أن يكتب بالصيغة الثنائية الرقمية **Binary** فقط الصفر والواحد، هذه الصيغة غير مناسبة لفهم بالنسبة للبشر ويعصب تفسيرها. لذا تقوم الشركات المصنعة للمعالجات والمُتحكمات الدقيقة بصناعة بعض الأدوات البرمجية التي تُسهل على المطوريين أن يصنعوا برامج بلغات مفهومة وقابلة للقراءة.

في البداية كانت الشركات تصمم برمجيات التجميع **Assemblers** التي توفر المطور مجموعة من الأوامر تسمى بأوامر التجميع **Instructions**.

والتي كانت أوامر قصيرة وسهلة نسبياً مثل **ADD** (أجمع رقمين) أو **SUB** (اطرح رقمين)، ولكن كان هناك عيوب كثيرة لكتابة البرامج بهذه اللغة مثل الحجم والوقت، حتى أن بعض البرامج كانت تصل إلى عشرات الآلاف من السطور. وكان هناك مثل شهير يقول "كتابة برنامج معقد بلغة الأسمبلية موازي لحرف أساسات ناطحة سحاب باستخدام ملعة".



ظل الأمر هكذا فترة من الزمن حتى ظهرت اللغات عالية المستوى **High level language** مثل لغة السي. وهي لغات تسهل كتابة الكود البرمجي وتحويله إلى لغة الآلة تلقائياً عن طريق المترجمات **Compilers** وبذلك أصبحت عملية تطوير الكود أسهل بكثير.

يستخدم لغة السي يمكننا تطوير برامج المُتحكمات الدقيقة كالتالي:

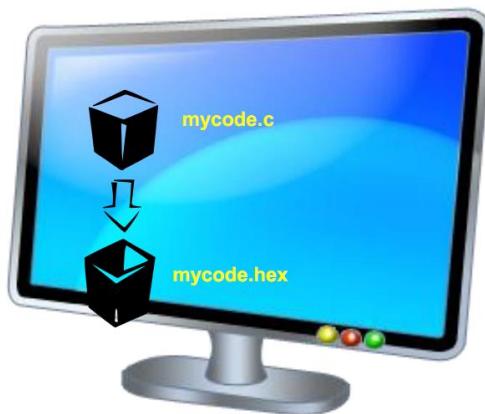
كتابه البرنامج بلغة السي: في هذه المرحلة نستخدم لغة السي للتغيير عن الوظائف التي نريد تنفيذها من المتحكم الدقيق

2. **توليد ملف الـ Hex :** ملف الـ hex هو الملف الذي يحتوي على البرنامج الحقيقي الذي سيخزن داخل ذاكرة المتحكم ويتم توليدة تلقائياً من تحويل الكود المكتوب بلغة السي إلى الأوامر البرمجية بصيغة hex عن طريق الـ toolchain (ستتحدث عنها بالفصيل في الفصل التالي).

من	البرنامج
إلى	الحاسوب
هذه	المُتحكم:
المرحلة	التي يتم
يسميها	(أو كما
حرق	بعملية
البيانات	(burn
داخل	الرفقية
ليبدأ	المُتحكم
بتنفيذها	حيث يقوم

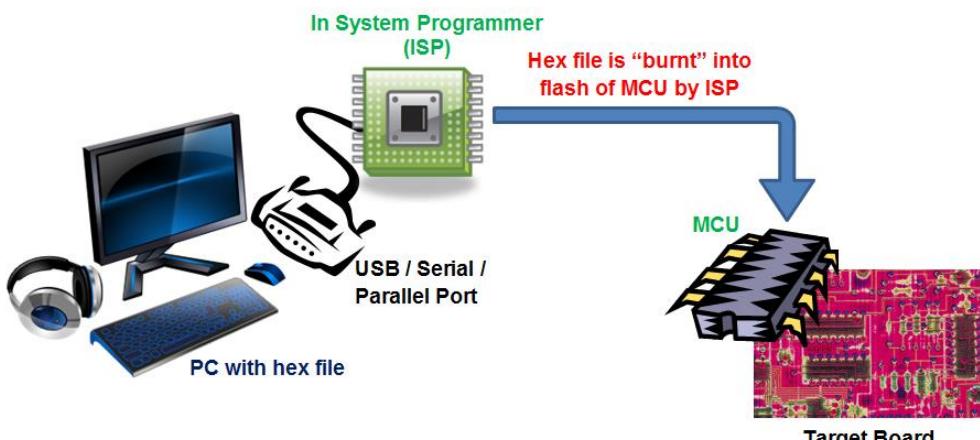
```
#define F_CPU 16000000UL
#include "servo.h"
int main( void )
{
    cli();
    Init_servo_out();
    sei();

    for ( ; )
    {
        //servo_channel_A_updated = 1;
    }
}
```



رفع
ذاكرة
كتابه
البعض
ذاكرة
برنامج

٤. البرنامج اختبار واكتشاف الأخطاء: في هذه المرحلة يتم تشغيل المتحكم في الواقع على لوحة التجارب أو على Test Kit للتأكد من أن البرنامج ينفذ المطلوب أو لاكتشاف أي أخطاء، وقد يتم تكرار هذا الأمر عشرات المرات حتى نصل إلى برنامج يؤدي الجميع الوظائف المطلوبة منه بأقل نسبة خطأ.



للفيام بكل ما سبق سنحتاج لمجموعة من الأدوات البرمجية والمكونات الإلكترونية وهو ما سيتم شرحه بالتفصيل في الفصل التالي.



ثانياً: مراحل تطوير العتاد

لتطوير أي مشروع سنحتاج أن نوصل المتحكم الدقيق بالمكونات الإلكترونية التي سيتحكم بها وهو ما يعرف بمفهوم الـ **Devices Interfacing** (واجهة الأجهزة المختلفة) فالتحكم الدقيق لا يعمل بمفرده وإنما يحتاج أجهزة أخرى لاستقبال منها قراءات (مثل الحساسات Sensors) أو ليتحكم بها مثل الشاشات والمحركات.

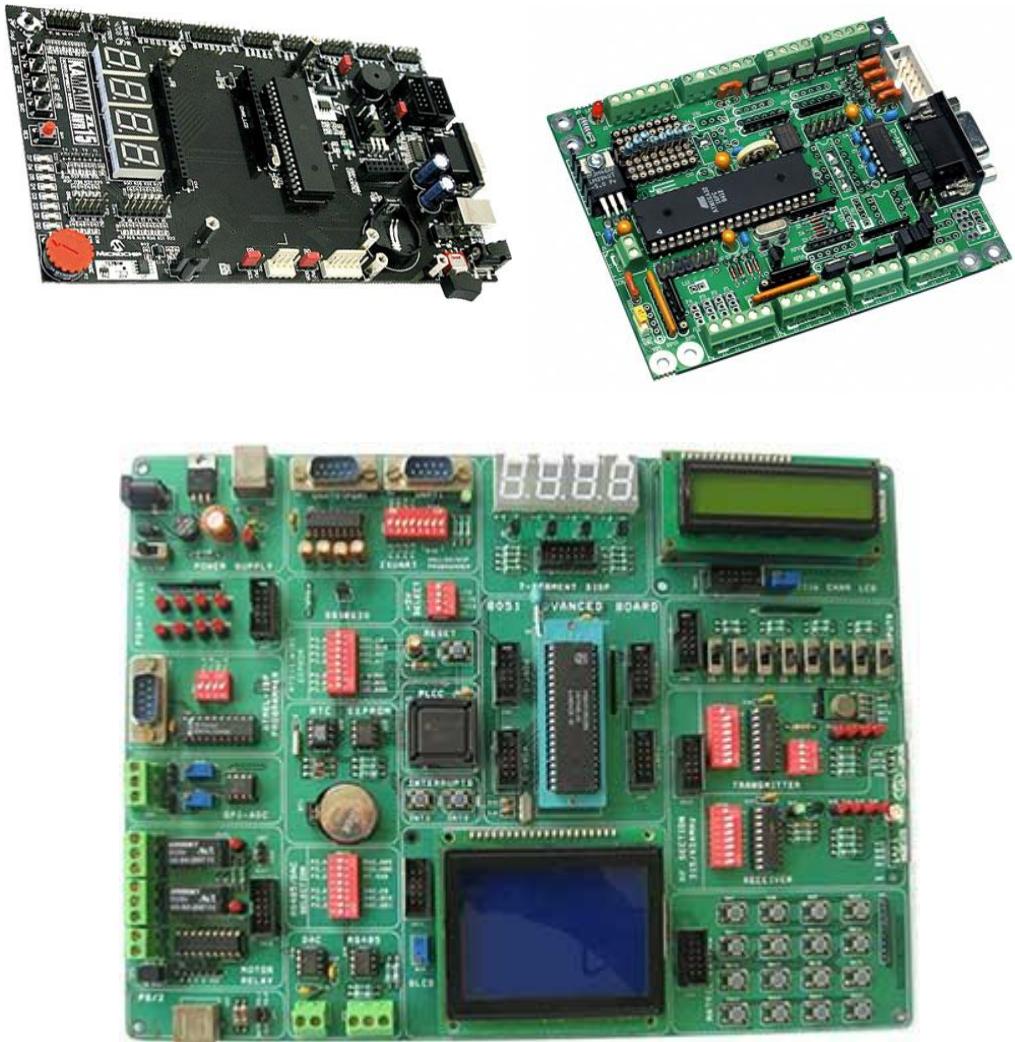
هناك طريقتين أساسيتين لعمل ذلك وهما، استخدام اللوحات التطويرية Development Kit أو استخدام لوحة التجارب Breadboard، كل طريقة لها مميزات وعيوب.

Development Kit



هي لوحة من المتحكم مجموعة العناصر المتصلة به جاهزة LCDشاشة، أزرار تحكم، حرارية Relay الاتصال محول يوجد بها أكثر ذلك. هذه
 اختبار مكونة + كبيرة من الإلكترونية بصورة التشغيل مثل لوحة مفاتيح، حساسات موضوعية، وبعض أدوات الرقمية مثل RS232 وقد أو أقل من اللوحات
 تُسهل عملية التطوير بصورة كبيرة فيبي تحتوي على معظم ما قد تحتاجه على لوحة واحدة جاهزة ومتصلة ببعضها البعض وبالتالي لن تحتاج لشراء مكونات أخرى أو توصيل عناصر إضافية وستوفر عليك وقت بناء الدوائر الإلكترونية.

الصور التالية هي لمجموعة مختلفة من لل Development kits



مرفق مع الكتاب مجلد يحتوي على تصميمات لمجموعة لوحات تطويرية مفتوحة المصدر مخصصة لـ AVR، أغلبها مصمم للمتحكم الدقيق ATmega32/ATmega16 و يمكنك صناعتها بنفسك بتكلفة أقل من شرائها.

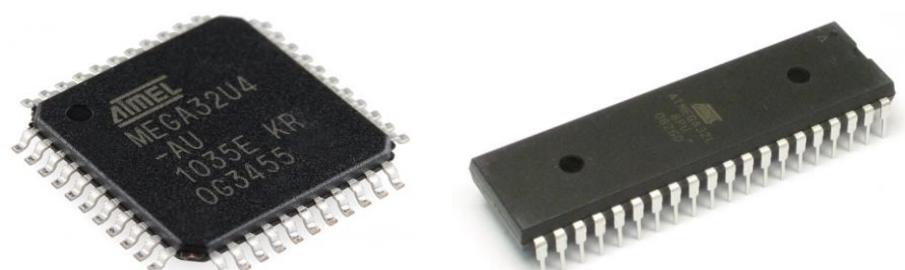
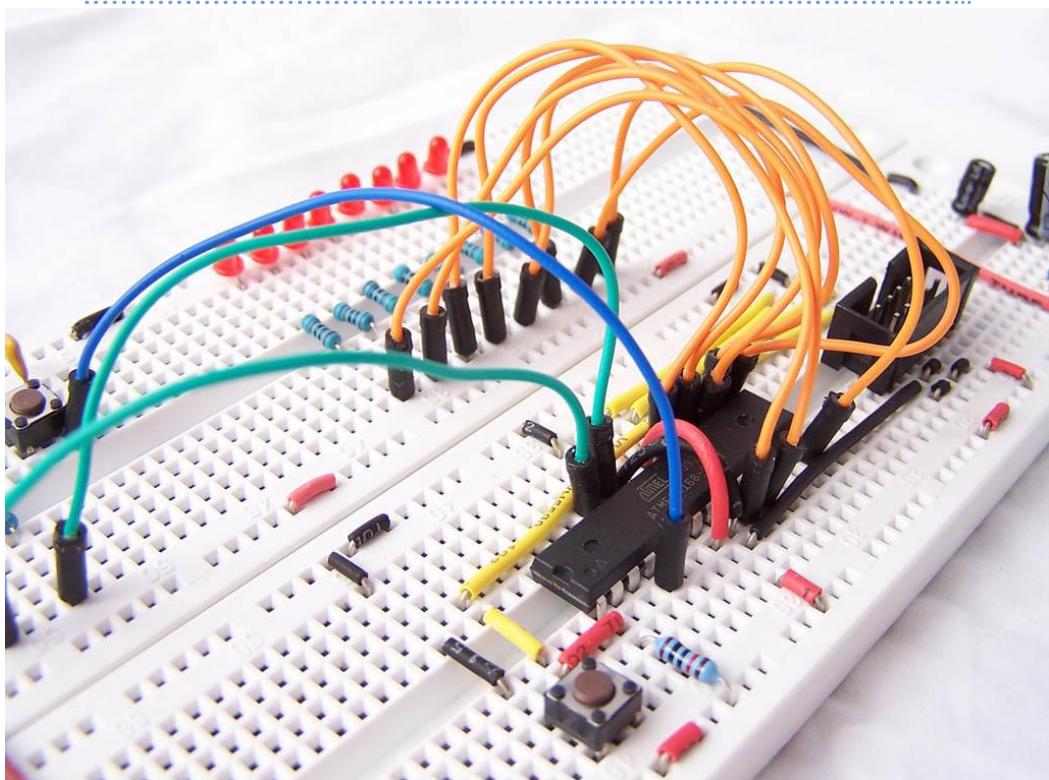
لوحة التجارب Breadboard

الطريقة الثانية هي استخدام لوحة التجارب البلاستيكية والتي تساعدهك على بناء أي دائرة إلكترونية باستخدام الأساند، تتميز هذه اللوحة بأنه يمكنك بناء أي دائرة قد تخطر على بالك فمن السهل أن تفك وتركب أي عنصر أو شريحة إلكترونية (من نوع DIP) على هذه اللوحة، سيتم استخدام هذه الطريقة في الكتاب لأنها الخيار الأرخص والأكثر توافراً في كل البلاد العربية.



إضافية: الإلكترونية
اللائحة
in-line
هي التي
من الأرجل
والتي يمكن
بتقويب على
التجارب أو
هي SMD
كلمة
Mount
اللائحة
الحجم و
معدنية
ويتم لحامها
وPCB فقط

معلومات
اللائحة
DIP (Dual
Package)
تمتلك صفين
المعدنية
توصيلها
لوحة
الـ PCB
اختصار
Surface
هي Device
صغيرة
تمتلك أرجل
صغيرة جداً
على سطح

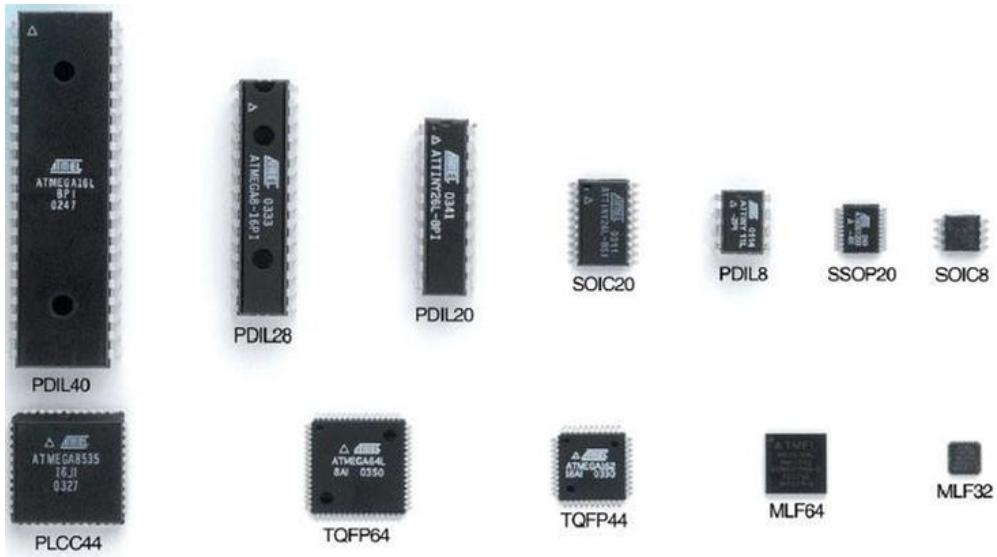


ATmega32 (DIP)



صور مختلفة
متحكمات AVR
وأحجام الـ

لأنواع تغليف
بجميع أنواع
SMD وـ DIP



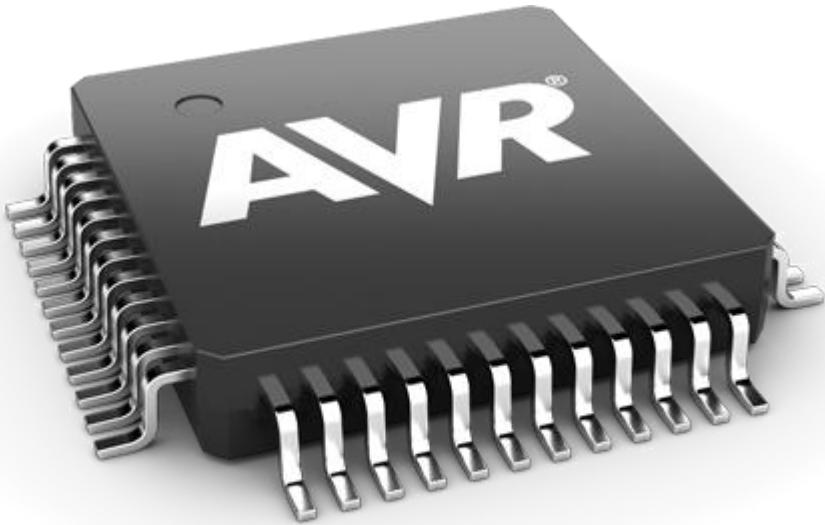


صفحة جديدة



2. نظرة عامة على مُتحكمات AVR

- ✓ تركيب المُتحكم الدقيق
- ✓ مميزات معمارية AVR
- ✓ كيف تختار المُتحكم المناسب من عائلات AVR المختلفة
- ✓ مقدمة



- عن قراءة دليل البيانات Datasheet
- نظرة عامة على المُتحكم ATmega16
- نظرة عامة على المُتحكم ATTiny84

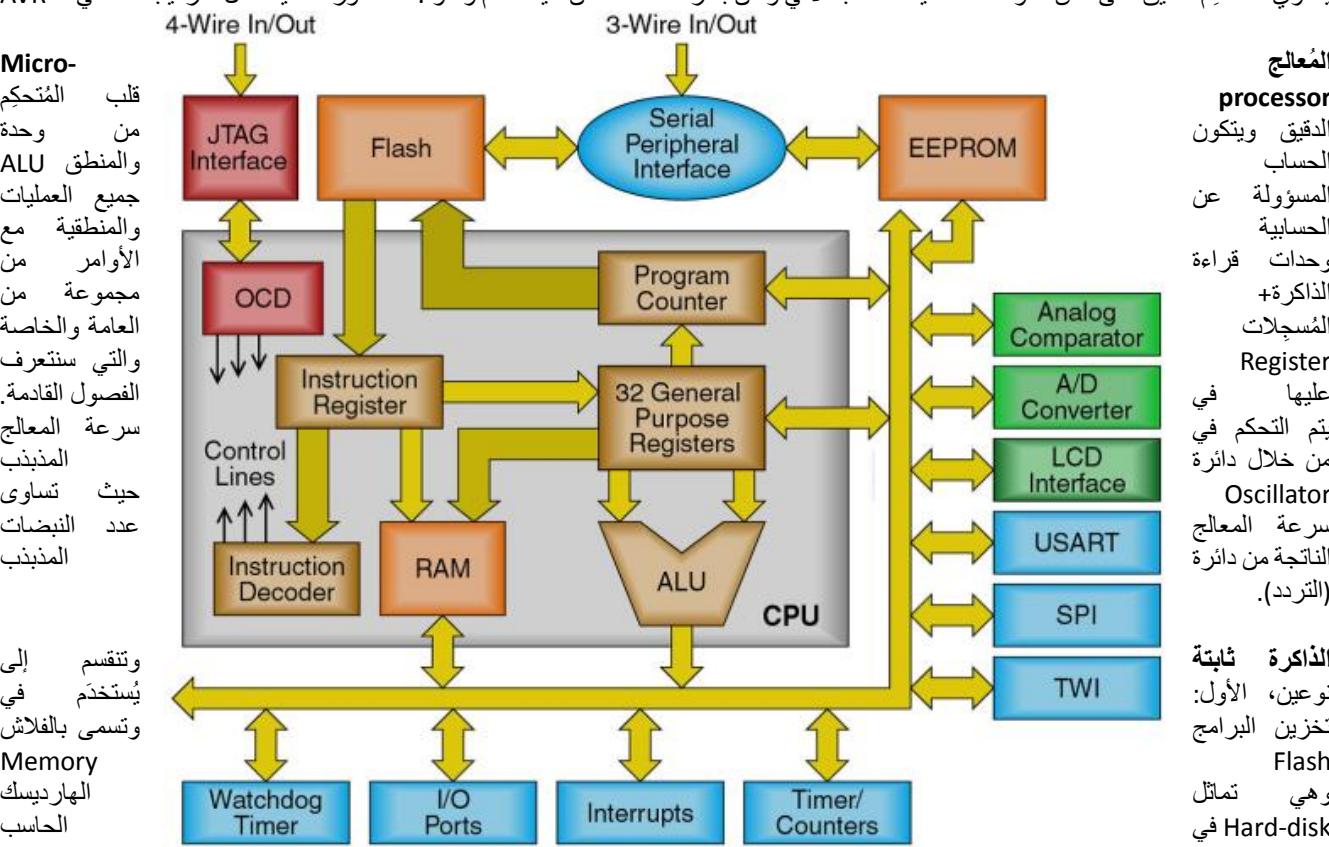
✓

✓



2.1 تركيب المُتحَكِّم الدقيق وعمارية AVR

المتحكم الدقيق هو حاسوب متكامل على شريحة واحدة Computer On Chip يُستخدم في التحكم بمجموعة من الأجهزة الأخرى. ومثل جميع الحواسيب تحتوي المتحكم الدقيق على نفس مكونات الداخلية للحاسوب الآلي ولكن بقدرات مختلفة من حيث القدرة والقوة. الصورة التالية تمثل التركيب الداخلي لـ AVR



Program memory. أما النوع الثاني فتسمى الـ ROM. إن يسمىها المتغيرات التي تؤثر في برنامج المُتحكّم ويجب أن تحفظ من الصياغ.

الذاكرة مؤقتة **SRAM** وهي اختصار لـ **Static RAM**، وهي تماثل الذاكرة العشوائية الموجودة في الحواسيب الشخصية التي نستخدمها.

المُسجّلات Registers وهي أحد صور الذاكرة وتماثل الذاكرة المؤقتة من حيث التركيب وطريقة العمل ولكنها تستخدم في التحكم بجميع إعدادات ووظائف المُتحكم الدقيق كما سنرى في الفصول القادمة تبعاً.

الوحدات الطرفية (sets) هي مجموعة من الوحدات التي تساعد المتحكم الدقيق في أداء وظيفته الأساسية أخرى من أمثل هذه الوحدات، المنفذ العامة PORTS، المحول التنازلي الرقمي ADC، الموقتات Timers، وحدات الاتصال ومعالجات مثل USART، SPI، i2C و بعض متحكمات AVR قد تحتوي على أنظمة للتشغير مدمجة بداخلها **CryptoAuthentication** .. الخ.

ما الفرق بين الذاكرة العشوائية (المؤقتة) داخل الحواسيب الشخصية والمتحكمات الدقيقة ولماذا تسمى Static وليس Dynamic؟ الفرق الأساسي هو العنصر الذي تصنع منه الذاكرة، حيث تتميز ذاكرة المتحكمات بأنها تصنع من الـ Flip-Flop الذي يتميز بالقدرة على الاحتفاظ بالبيانات بأقل تيار كهربائي ممكن وبالتالي فهو الخيار الأفضل من ناحية الاستهلاك للطاقة كما أن البيانات الموجودة عليه لا تحتاج لعملية تجديد Refreshing مثل الذاكرة "dynamic RAM" الموجودة في الحواسيب القديمة والتي تصنع من المكثفات الطفيلي Parasitic Capacitors والتي



تحتاج دائماً لعملية تجديد Refreshing ، والـ Refreshing يقتضي بداخلها مع مرور الوقت (أكثر من 10 ملي ثانية كثيلة بداخلها مع مرور الوقت) لاستخدام البيانات من DRAM ، كما أنها تستهلك الكثير من الطاقة بسبب هذه العملية.

ما الفرق بين الـ **Flash** والـ **EEPROM** بالرغم أن كلاهما تقنياً يعتبر **EEPROM**؟

الـ **EEPROM** هي اختصار لعبارة **Electrical Erasable Read Only Memory** والتي تعني الذاكرة التي تستخدم القراءة فقط ويتبرجمتها كهربياً.

المتحكمات الدقيقة غالباً ما تحتوي على نوعين من الـ **EEPROM** الأولى تسمى الـ **Flash** لأنها سريعة جداً في كتابة البيانات وقد تصل سرعة الكتابة عليها إلى واحد **MegaBit/S** ، فمثلاً قد تكتب 1 بait بداخل الفلاش في زمن 1 ميكروثانية فقط. بينما الـ **EEPROM** التقليدية بطبيعة مقارنة بالفلاش حيث أن كتابة 1 بait بداخلها قد يستغرق 1 ملي ثانية (يعني أبطأ بنحو 1000 مرة من الـ **Flash**).

2.2 مميزات معمارية الـ **AVR**

يقصد بكلمة معمارية **Architecture** طريقة توصيل المكونات الداخلية للمتحكم مع بعضها البعض ومدى حجم البيانات التي تستطيع هذه المكونات أن تعالجها. فمثلاً جميع متحكمات AVR يوجد بها المكونات السالبة ذكرها وبينها العديد من الأشياء المشتركة، لكن سنجده أنه هناك اختلافات رئيسية بين العديد من متحكمات الـ **AVR**.

بعض الخصائص المشتركة بين جميع متحكمات AVR

- معمارية **Harvard** هذه المعمارية الحديثة نسبياً تعني أن المعالج المركزي يستطيع أن يتواصل مع الذاكرة **RAM** والـ **ROM** في نفس الوقت حيث نجد أن جميع متحكمات الـ **AVR** تستطيع أن تكتب في الـ **RAM** وتقرأ من الـ **ROM** في نفس اللحظة. على عكس المعماريات القديمة مثل **Von Neumann** والتي تسمح للمعالج أن يقوم بعمل شيء واحد فقط (إما القراءة أو الكتابة في نفس اللحظة).

- معظم متحكمات AVR تمتلك القدرة على تنفيذ أوامر برمجية = سرعة المعالج فمثلاً إذا كان تردد المعالج = 16 ميجا (16 مليون نبضة) فهذا يعني أن المتحكم يستطيع أن ينفذ 16 مليون أمر في الثانية الواحدة. ويرجع الفضل إلى وجود نسختين من أنظمة قراءة الذاكرة وفك تشفير الأوامر $2 \times \text{program counter} + 2 \times \text{instruction decoder}$ تعملان معاً في نفس الوقت مما يضاعف سرعة وعدد الأوامر التي يتم نسخها من الذاكرة.

- تختلف المُتحكمات فيما بينها على حسب الـ **peripheral units** الموجودة بداخلها وتقنية معالجة البيانات سواء كانت 8 أو 16 أو 32 بت. وتعتبر أحد مميزات الـ **AVR** الراة وهي إمكانية استخدام الذاكرة الفلاش لتخزين المتغيرات أثناء تشغيل المُتحكم (كأنها تقوم بوظيفة الـ **EEPROM** التقليدية) حيث يمكن استخدام بعض الأوامر البرمجية لتغيير محتوى الـ **Flash memory** أثناء تشغيل المُتحكم وبدون استخدام أي مبرمج خارجية (burner) **Programmer**. يمكنك معرفة كافة التفاصيل من الملف الذي أصدرته شركة **ATmel** ويشرح جميع الأوامر البرمجية لهذه الميزة الراة ([بلغة السي](http://www.ATmel.com/Images/doc2575.pdf)). <http://www.ATmel.com/Images/doc2575.pdf>

يعبر هذا الرقم عن حجم البيانات الذي يستطيع المعالج المركزي **CPU** داخل المُتحكم الدقيق أن يتعامل معه في النبضة الواحدة.

فمثلاً إذا كان المُتحكم من نوع 8 بت فإنه يستطيع أن يجمع رقمين 8 بت مع بعضهم في نبضة الواحدة. لكن إذا جعلت المعالج يجمع رقمين بطول 16 بت فإنه سيضطر أن يتعامل مع الأرقام على أكثر من مرة بحيث يجزأ الأرقام إلى مجموعات 8 بت فقط. أما المُتحكمات الـ 32 بت تعني أن المعالج يمتلك القدرة على القيام بجميع العمليات الحسابية والمنطقية على بيانات بطول 32 بت في النبضة الواحدة.

ما معنى 8 بت أو 32 بت؟



2.3 كيف تختار بين عائلات AVR المختلفة

تعتبر مهارة اختيار المُتحكم المناسب من أهم ما يجب أن يتعلم أي مهندس نظم مدمجة. حيث أن الشركات المنتجة للمتحكمات الدقيقة عادة ما تصنع المئات من المُتحكمات الدقيقة وتقسمها إلى عائلات تختلف فيما بينها على حسب السعر والإمكانيات لكل مُتحكم. لذا سيتوجب عليك أن تتقن اختيار المُتحكم المناسب لأداء أفضل تصميم بأقل سعر ممكن.

تعتبر أهم العوامل المؤثرة في تصنیف المُتحكمات الدقيقة هي:

سرعة معالجة البيانات والإستجابة	عدد أطراف التحكم العامة	مساحة الذاكرة المطلوبة والتي سيعمل بها المُتحكم	الـ Peripheral Devices المتوفرة	عدد وامكانيات المُتحكم	الـ GPIO والتي تمثل عدد المدخل والمخرج
المطلوبة	ال العامة	الـ المطلوبة	الـ المتوفرة	الـ المطلوبة	الـ المطلوبة

على حسب العوامل الخمسة السابقة سنجد أن شركة ATmel قسمت مُتحكمات AVR إلى 6 عائلات أساسية منها أربعة عائلات عامة General purpose microcontrollers مما يعني أنه يمكن استخدامها لجميع مجالات النظم المدمجة ومختلف المنتجات. وهناك عائلتين مصممتين لمنتجات محددة فقط:

العائلات العامة General purpose microcontrollers

megaAVR – 8 bit

ATTiny – 8 bit

bit16 & AVR Xmega – 8 bit

AVR - 32 bit

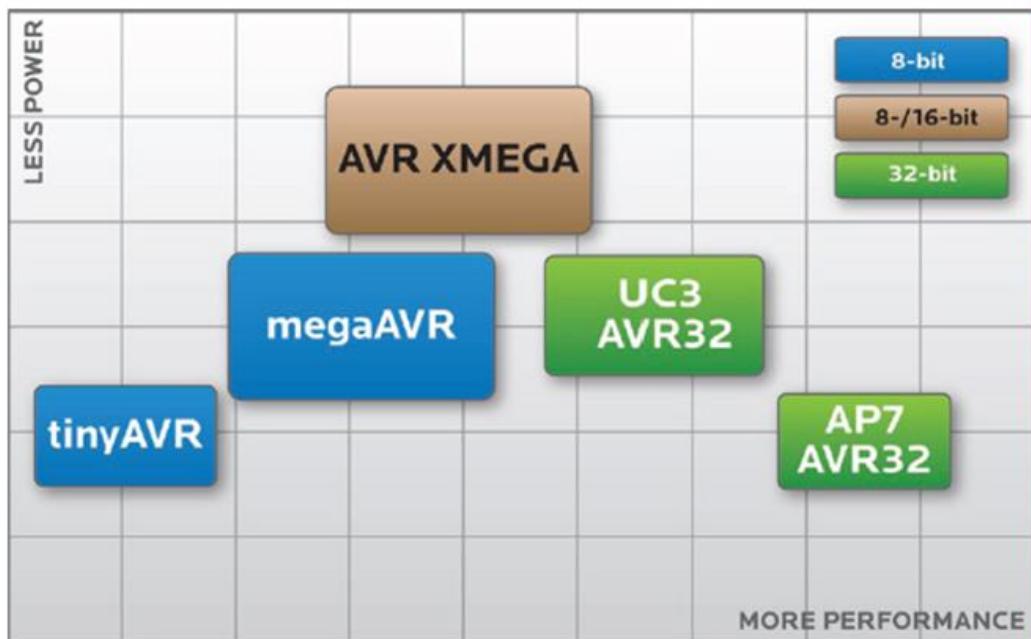


العائلات المتخصصة Special purpose microcontrollers

هذه العائلة مصممة لأنظمة المدمجة الخاصة بالسيارات لذا فهي تتميز بتحمل الظروف القاسية مثل درجات الحرارة المرتفعة (يمكنها العمل في بيئة تصل درجة حرارتها إلى 150 درجة مئوية)، كما تحتوي على نظم تشفير خاصية لحماية المعلومات المخزنة بداخلها وكذلك تحتوي على أنظمة حماية من مشاكل التيار الكهربائي وفرق الجهد (مثل حدوث قصر في الدائرة short circuit).

عائلة المُتحكمات الخاصة بإدارة وتشغيل البطاريات، ومصممة لتدير عملية الشحن والتغذية الآمن للبطاريات **Battery Management** سيرتكز الكتاب على شرح العائلة الأولى والثانية mega, ATTiny باعتبارهم أشهر العائلات وأكثرها توافراً على مستوى العالم

الصورة التالية توضح ترتيب القوة والميزات التي تحتويها كل عائلة، حيث نجد أن الخط الأفقي يعبر عن قوة الأداة Performance والخط الرئيسي يعبر عن مدى انخفاض استهلاك الطاقة.



عند الضغط على أي من أسماء العائلات بالأعلى ستنتقل إلى صفحة على موقع Atmel توضح جميع أفراد هذه العائلة من المُتحكمات مع عرض سريع لخصائص كل مُتحكم مثل حجم الذاكرة وعدد أطراف المُتحكم. هذه الصفحة تقدم مقارنة سريعة بين المُتحكمات لتساعدك على اختيار المُتحكم الأنسب لمشروعك.

عند الضغط على اسم أي من المُتحكمات مثل ATmega16 سيتم نقلك إلى صفحة المُتحكم والتي تحتوي على جميع البيانات المتعلقة بهذا المُتحكم بما في ذلك أهم ملف وهو "الدليل البياني Datasheet" والذي يتوفّر منه نسختين، "مختصر سريع" أو الدليل الكامل complete summary.



Device	Description
ATmega168PB	8-bit AVR Microcontroller, 16KB Flash, 32-pin
ATmega48	8-bit AVR Microcontroller, 4KB Flash, 28/32-pin
ATmega48A	8-bit AVR Microcontroller, 4KB Flash, 28/32-pin
ATmega48P	8-bit picoPower AVR Microcontroller, 4KB Flash, 28/32-pin
ATmega48PA	8-bit picoPower AVR Microcontroller, 4KB Flash, 28/32-pin
ATmega48PB	8-bit Atmel® AVR® Microcontroller, 4KB Flash, 32-pin
ATmega88PB	8-bit Atmel® AVR® Microcontroller, 8KB Flash, 32-pin
ATmega8	8-bit AVR Microcontroller, 8KB Flash, 28/32-pin

في هذا الكتاب دائماً سنستخدم الدليل الكامل من أي **Datasheet**. لذا قم بتحميل كل من الدليل الخاص بالمتحكم **ATtiny84** و **ATmega16**

من خلال الصفحات السابقة وملفات الـ **Datasheet** يمكنك تحديد المتحكم الذي يمتلك الإمكانيات المناسبة للمشروع الذي تريده. بالتأكيد اختيار المتحكم يجب أن يكون مقترباً من خبرتك في مجال البرمجة وتحسين الأكواد المكتوبة للاستفادة القصوى من المتحكم. لذا سنجد أن مهارة اختيار المتحكم المناسب سترداد عندما تتفق برمجة هذا النوع من المتحكمات.



2.4 قراءة دليل البيانات Datasheet

تساعدك Datasheet على فهم المتحكم الدقيق بصورة مفصلة فهي تحتوي على طريقة تشغيله وبرمجته، وتحتوي أيضاً على جميع البيانات التقنية المتعلقة بالتحكم مثل: التصميم الداخلي، وظائف الأطراف، المسجلات، الطاقة، تقنيات البرمجة، كيفية تفعيل القدرات التي يملكها المتحكم أو إلغائها ... الخ. وتعتبر المرجع الشامل لأي متحكم.

ستتناول الم الموضوعات المختلفة في دليل البيانات على مدار فصول الكتاب بالكامل، حيث سنتعلم في كل فصل أحد الخصائص التي تتمتع بها متحكمات AVR وسنحصل على تفاصيل هذه الخصائص من دليل البيانات.

هذا الفصل سيركز على الجزء الأول من دليل البيانات والذي غالباً ما يكون أول 5 أو 8 صفحات ويحتوي على النظرة العامة للمتحكم.

Features

- High-performance, Low-power Atmel® AVR® 8-bit Microcontroller
- Advanced RISC Architecture
 - 131 Powerful Instructions – Most Single-clock Cycle Execution
 - 32 x 8 General Purpose Working Registers
 - Fully Static Operation
 - Up to 16 MIPS Throughput at 16 MHz
 - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory segments
 - 16 Kbytes of In-System Self-programmable Flash program memory
 - 512 Bytes EEPROM
 - 1 Kbyte Internal SRAM
 - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
 - Data retention: 20 years at 85 °C/100 years at 25 °C^[1]
 - Optional Boot Code Section with Independent Lock Bits
 - In-System Programming by On-chip Boot Program
 - True Read-While-Write Operation
 - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
 - Boundary-scan Capabilities According to the JTAG Standard
 - Extensive On-chip Debug Support
 - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
 - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
 - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
 - Real Time Counter with Separate Oscillator
 - Four PWM Channels
 - 8-channel 10-bit ADC



8-bit AVR® Microcontroller with 16K Bytes In-System Programmable Flash

ATmega16
ATmega16L



ATmega16/ATmega32 الخصائص العامة للمُتحكِّم 2.5

في الصفحة الأولى من دليل البيانات نجد المعلومات المتعلقة بالخصائص العامة للمُتحكِّم (مع العلم أن هذه الخصائص تكون مشتركة بين معظم أفراد العائلة من المُتحكِّمات وقد تختلف فيما بينها بفروقات بسيطة) وهي كالتالي:

المعمارية Advanced RISC (Harvard based) Architecture

- يوضح هذا الجزء الخصائص العامة لتقنية معالجة البيانات وسرعة المُتحكِّم الدقيق حيث نجد أن المُتحكِّم يتمتع بالقدرات التالية:
عدد 131 – 16 – **single cycle execution Instruction** والتي تعني أن المُتحكِّم يمكن برمجته باستخدام 131 أمر بلغة الأسمبلي ومعظم هذه الأوامر يتم تنفيذها في نبضة واحدة فقط.

- **Mega Instruction Per Second (MIPS) 16** وهي نفس الخاصية السابقة والتي تعني أن المُتحكِّم يمكنه تنفيذ 16 مليون أمر برمجي عندما يتم تشغيله بتردد 16 ميجاهرتز (هذا بسبب أن معظم الأوامر البرمجية يمكنه تنفيذها في نبضة واحدة فقط). وتعبر هذه الخاصية عن أقصى سرعة معالجة للمُتحكِّم الدقيق وتعتبر من أهم الخصائص التي تتمتع بها معالجات AVR.

- مثال: إذا كان لدينا برنامج مكون من 10 أوامر بلغة الأسمبلي والمُتحكِّم يعمل بسرعة 1 ميجاهرتز (مما يعني أن ز من كل نبضة = 1 ميكروثانية) فهذا يعني أن البرنامج سيستغرق تنفيذه ز من 10 نبضات وهو ما يساوي 10 ميكروثانية فقط.

- **On-Chip 2 cycle multiplier** في الأجيال القديمة من المعالجات والمُتحكِّمات الدقيقة كان يتم حساب عملية ضرب الأرقام باستخدام الجمع المتكرر فمثلاً حاصل ضرب $12 \times 10 = 120$ = جمع رقم 12 مع نفسه 10 مرات ($12+12+12+12+12+12+12+12+12+12$الخ). وهذا يعني تنفيذ أمر "الجمع" 10 مرات (بلغة الأسمبلي) وبالتالي تستغرق وقت = 10 نبضات. أما في مُتحكِّم AVR نجد وحدة معالجة الضرب تقوم بتنفيذ أي عملية ضرب في نبضتين فقط مما يسرع هذا النوع من العمليات الحسابية بصورة كبيرة.



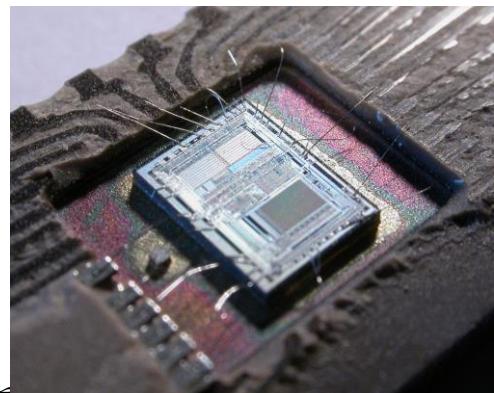
الذاكرة عالية التحمل *High endurance Memory*

يوضح هذا الجزء الخصائص التي تمتاز بها الذاكرة الموجودة داخل مُتحكمات AVR بأنواعها المختلفة مثل الـ Flash والـ RAM والـ EEPROM ومن أهم هذه الخصائص التالي:

- 16 كيلو بايت من الذاكرة الثابتة **Self-programmable Flash memory** والتي تستخدم لحفظ البرنامج الذي سيشغل المُتحكم الدقيق وتتمتع بخاصية البرمجة الذاتية التي تحدثنا عنها سابقاً. (يمتلك ATmega32 ذاكرة ثابتة 32 كيلوبايت).
- 512 بait (8 * 512 بait) من ذاكرة **EEPROM**
- 1 كيلوبايت من الذاكرة المؤقتة (العشوانية) **SRAM**
- إمكانية الكتابة (برمجة) / مسح محتوى ذاكرة الفلاش نحو 10,000 مرة
- إمكانية الكتابة (برمجة) / مسح محتوى ذاكرة EEPROM نحو 100,000 مرة.
- الاحتفاظ بالبيانات في كل من الـ Flash والـ EEPROM لمدة زمنية تصل إلى 100 عام كامل عند تشغيل المُتحكم في درجة حرارة 25 درجة مئوية أو 25 عام عند تشغيل المُتحكم في درجة حرارة 80 درجة مئوية وهذا يعني أن المُتحكم يستطيع العمل والاحتفاظ بالبيانات لفترة طويلة جداً.
- إمكانية استخدام **Bootloader** (التفاصيل مذكورة في فصل الفيوزات).
- **True Read-While-Write Operation** تعني أن المُتحكم يستطيع قراءة بيانات من الـ ROM بينما يقوم بكتابة بيانات في الـ RAM في نفس الوقت على عكس المُتحكمات القديمة والتي كانت تستطيع أن تقوم بأحدى هذه العمليات فقط في نفس اللحظة.
- طبقة من الحماية لمنع سرقة البيانات المخزنة على ذاكرة المُتحكم باستخدام الـ **Lockbits** (التفاصيل مذكورة في فصل الفيوزات).
- الأجزاء التالية من دليل البيانات مثل Itag, Peripheral devices, Power consumption سيتم شرحها في فصولها الخاصة.

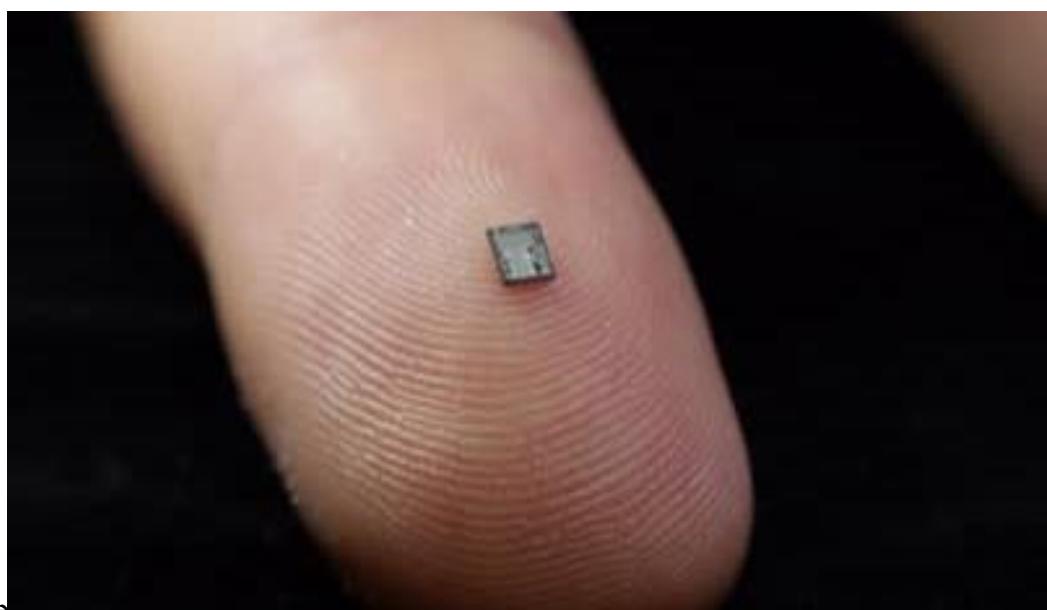


Microcontroller packaging



صورة توضح المتحكم الدقيق من الداخل والذي لا ينطوى حجمه (في معظم الحالات) أكثر من 10% من حجم الغلاف

الحجم الحقيقي لشريحة السيليكون التي يتكون منها المتحكم الدقيق غالباً ما يكون صغير جداً لدرجة أنه قد يصل إلى حجم "رأس عود ثقب" مما يجعل استخدامه مباشرة عملية صعبة، لذا يتم تصميم هيكل خارجي أكبر حجماً من مادة الـ Epoxy ويسمى الـ Packaging (الغلاف) للمتحكم الدقيق، ويخرج منه بعض الأطراف المعدنية الصغيرة التي تتصل بالمتحكم الدقيق الحقيقي.



حجم شريحة السيليكون مقارنة

بحجم إصبع الإنسان

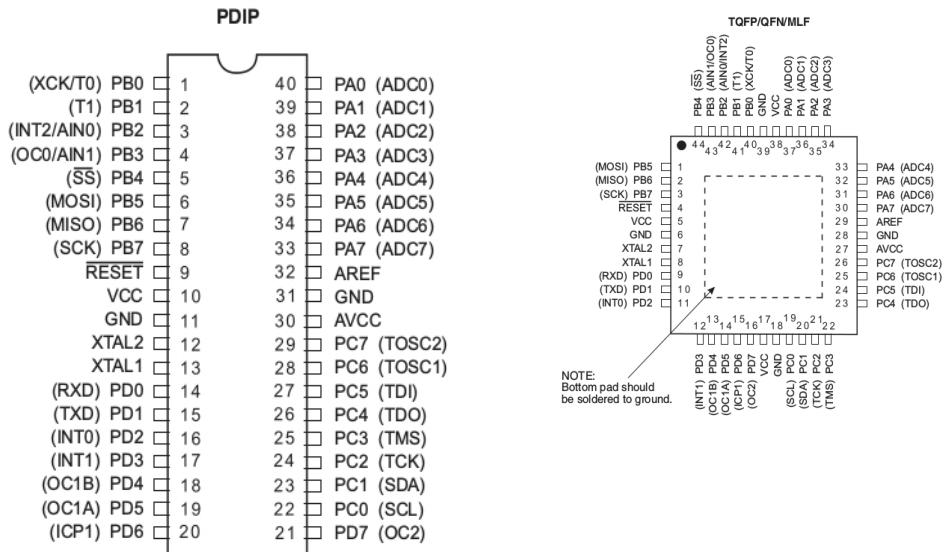
تتوفر المتحكمات الدقيقة بكل النوعين من التغليف DIP و SMD و سنجد أن شركة أتميل عادة ما تقوم بإصدار معظم المتحكمات من عائلة atmega وبكلا النوعين DIP و SMD فمثلا سنجد في الصفحة الثانية من دليل البيانات وحدات الـ package الخاصة بالمتحكم ATmega16 وهي attiny



DIP

SMD – TQFP

ملاحظة: الـ SMD يتوفر منه أحجام وأشكال مختلفة مثل TQFP, BGA, QFN, MLF



عندما تقوم الشركات بصناعة منتج ما فإنها تفضل استخدام معظم الشرائح الإلكترونية بتغليف SMD، حيث تتميز برخص السعر مقارنة بالـ DIP كما أنه يمكن تصميم دوائر تحتي على الكثير من المكونات بمساحة صغيرة جداً بفضل الحجم الصغير الذي تتمتع به شرائح الـ SMD.

ونجد نتيجة لهذا الأمر أن العائلات المطورة مثل AVR32 أو Xmega غالباً ما يتم إنتاجها بتغليف SMD فقط وذلك لأنها تحتوي على الكثير من الأطراف (قد يصل إلى 120 طرف) لذا سيكون من الصعب (والمكلف أيضاً) أن تصنع بتغليف DIP لأن الحجم سيكون ضخم جداً.



2.6 أطراف المُتحكِّم ATmega16

يتمكِّن المُتحكِّم ATmega16 ما مجموعه 40 طرف pin - موزعة على 4 بورتات كل واحد منها 8 أطراف وهم أطراف المُتحكِّم مجموعه من الأطراف المتعلقة بالطاقة والتردّد كما في الصورة التالية.

The diagram shows the pinout of the ATmega16 microcontroller. The pins are grouped into four ports:

- PORTB:** Pins 1-8. Includes XCK/T0 (PB0), T1 (PB1), INT2/AIN0 (PB2), OC0/AIN1 (PB3), SS (PB4), MOSI (PB5), MISC (PB6), and SCK (PB7).
- RESET:** Pin 9.
- VCC:** Pin 10.
- GND:** Pin 11.
- XTAL2:** Pin 12.
- XTAL1:** Pin 13.
- PORTD:** Pins 14-20. Includes RXD (PD0), TXD (PD1), INT0 (PD2), INT1 (PD3), OC1B (PD4), OC1A (PD5), and ICP1 (PD6).
- PORTA:** Pins 21-30. Includes PA0 (ADC0), PA1 (ADC1), PA2 (ADC2), PA3 (ADC3), PA4 (ADC4), PA5 (ADC5), PA6 (ADC7), PA7 (ADC7), AREF, GND, and AVCC.
- PORTC:** Pins 21-29. Includes PC7 (TOSC2), PC6 (TOSC1), PC5 (TDI), PC4 (TDO), PC3 (TMS), PC2 (TCK), PC1 (SDA), PC0 (SCL), and PC7 (OC2).

هذا الطرف يقف للّمُتحكِّم الدقيق ويعني أنه جميع المسجلات (يُجعل ويُعيد تشغيل البرنامج ذاكرة المُتحكِّم من البداية، طرف active low يعني عندما يتصل بالأرضي يحصل على إشارة يحصل أن يوصل دائماً طريق مقاومة 10 كيلو سيطر المُتحكِّم يقوم بعمل يشغل البرنامج المُخزن

هذا الطرف الذي يستقبل الموجب للّبطارية أو المستخدم (يُجب أن يكون من 2.7 فولت حتى 5.5 فولت حتى أقصى).

الطرف الأرضي للمُتحكِّم بالطرف الأرضي الكهرباء مصدر نتساءل لما يمتلك المُتحكِّم الأرض؟ السر

الطرف RESET يعمل سيعيد تنصير قيمتها ببصفر) الموجود في مع العلم أنه يتم تفعيله أو GND LOW logic بالـ VCC عن VCC أو (وإلا RESET ولن بالذاكرة أبداً).

الطرف VCC الطرف مصدر الطاقة موجب بداية فولت بـ

الطرف GND ويتم توصيله البطارية أو المستخدم. قد زوج من

هو تقليل الضجيج الكهربائي Noise، فعندما يتواجد أكثر من مسار للأرضي فإن ذلك يحسن في القضاء على noise خاصة إذا كان المُتحكِّم يقوم بـ توليد إشارات عالية السرعة (بالميجاهرتز).

الطرفان XTAL1, XTAL2 الأطراف التي يتم توصيلها بدائرة المذبذب الخارجي والتي سنتعرف على جميع أنواعها بالتفصيل في فصل (الفيوزات)، سرعة التشغيل والطاقة).

الطرف AVCC اختصار لـ VCC هذا الطرف مسؤول عن تشغيل المحول التناظري الرقمي ADC الموجود داخل المُتحكِّم ويجب أن يتم توصيله دائماً بنفس الجهد الذي يتصل به الـ VCC.

الطرف AREF اختصار لـ Analog Reference والذي سيتم شرحه بالتفصيل مع الـ ADC.

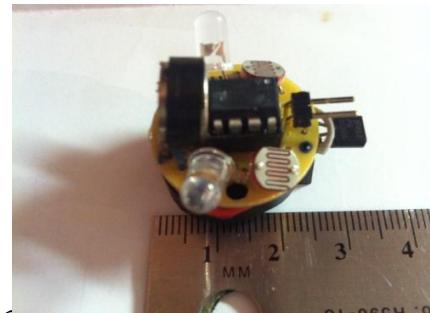
باقي أطراف المُتحكِّم موزعة على الـ 40 بورتات المختلطة A,B,C,D والتي تمتلك القدرة على التحكم بالمكونات الإلكترونية المختلفة كما تستطيع استقبال البيانات القادمة من الحساسات (سواء كانت رقمية أو تماثيلية) لذا تسمى "منفذ إدخال/إخراج عام" GPIO كما تمتلك مجموعة من الوظائف الإضافية



مثل الاتصالات التسلسلية، المقاطعات .. الخ، والتي سنتعرف عليها تبعاً في الفصول التالية.

2.7 عائلة ATTiny

تعتبر هذه العائلة من المُتحكمات حديثة نسبياً وهي مخصصة للاستخدامات التي تحتاج مُتحكم دقيق صغير الحجم وسريع في ذات الوقت دون التضحية بالميزات الكاملة التي قد تجدها في معظم المُتحكمات لذا قامت شركة Atmel بإنتاج هذا الجيل المتميز من المُتحكمات والمعروفة باسم ATTiny اختصاراً لعبارة Atmel Tiny.



صورة روبوت صغير باستخدام المُتحكم ATTiny85 عندما ننظر للصفحة الأولى من دليل البيانات للمُتحكمات ATTiny 45/84/85.

هذا المُتحكمات تمتلك معظم القدرات الموجودة في سلسلة megaAVR فهي تمتلك نفس المعمارية وتعمل بسرعات تصل إلى 20 ميجاهرتز مع القدرة على تنفيذ 20 مليون أمر برمجي في الثانية الواحدة (هذا يعني أنها تتفوق على المُتحكمات الأقدم نسبياً مثل ATmega16/32/128).

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
 - 120 Powerful Instructions – Most Single Clock Cycle Execution
 - 32 x 8 General Purpose Working Registers
 - Fully Static Operation
- Non-Volatile Program and Data Memories
 - 2/4/8K Bytes of In-System Programmable Program Memory Flash
 - Endurance: 10,000 Write/Erase Cycles
 - 128/256/512 Bytes of In-System Programmable EEPROM
 - Endurance: 100,000 Write/Erase Cycles
 - 128/256/512 Bytes of Internal SRAM
 - Data Retention: 20 years at 85°C / 100 years at 25°C
 - Programming Lock for Self-Programming Flash & EEPROM Data Security
- Peripheral Features
 - One 8-Bit and One 16-Bit Timer/Counter with Two PWM Channels, Each
 - 10-bit ADC
 - 8 Single-Ended Channels
 - 12 Differential ADC Channel Pairs with Programmable Gain (1x / 20x)
 - Programmable Watchdog Timer with Separate On-chip Oscillator
 - On-chip Analog Comparator
 - Universal Serial Interface
- Special Microcontroller Features
 - debugWIRE On-chip Debug System
 - In-System Programmable via SPI Port
 - Internal and External Interrupt Sources: Pin Change Interrupt on 12 Pins
 - Low Power Idle, ADC Noise Reduction, Standby and Power-Down Modes



8-bit AVR®
Microcontroller
with 2/4/8K
Bytes In-System
Programmable
Flash

ATTiny24
ATTiny44
ATTiny84

الأساسي بين

الفارق
هذه العائلة

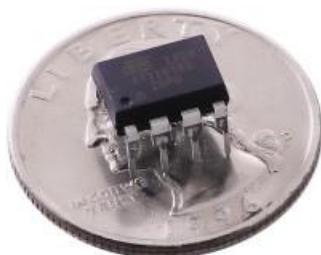
megaAVR هو الحجم وعدد الأطراف التي تعمل كGPIO حيث نجد أصغر مُتحكم في هذه العائلة يملك 6 أطراف فقط منهم 4 GPIO و 2 للطاقة مثل ATTiny4/5/9/10. و**تتميز مُتحكمات هذه العائلة بالقدرة على العمل بفرق جهد 1.8 فولت.**



مجموعة من أصغر المُتحكمات من شركة أتمل مقارنة

بعملة معدنية وتسمى *ATTiny10*

بالرغم من الحجم الصغير جداً إلا أن هذه المُتحكمات تتضمن معظم المُتحكمات الموجدة في Peripheral Devices فمثلاً المُتحكم *megaAVR* يمتلك 6 أطراف تحكم فقط إلا أنه يمكن تشغيلها كـ *GPIO, ADC, PWM, SPI*.



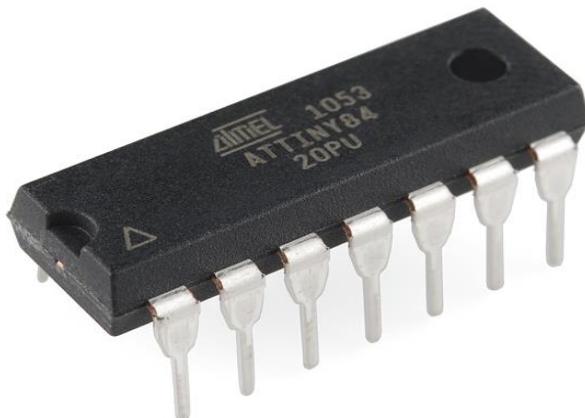
ATTiny45 / ATTiny85	
Reset	<input type="checkbox"/> 1
(Analog Input 3) Pin 3	<input type="checkbox"/> 2
(Analog Input 2) Pin 4	<input type="checkbox"/> 3
(-) GND	<input type="checkbox"/> 4
	<input type="checkbox"/> 5
	<input type="checkbox"/> 6
	<input type="checkbox"/> 7
	<input type="checkbox"/> 8
	VCC (+)
	Pin 2 (Analog Input)
	Pin 1 (PWM, MISO)
	Pin 0 (PWM, AREF,

سنستخدم في التجارب القادمة المُتحكم *ATTiny84* (بجانب المُتحكم *ATmega16*) والذي يمتلك 12 طرف تحكم و 2 طرف للطاقة.



ATtiny44 / ATtiny84

(+) VCC	1	14	GND (-)
Pin 10	2	13	Pin 0 (Analog Input 0, AREF)
Pin 9	3	12	Pin 1 (Analog Input 1)
Reset	4	11	Pin 2 (Analog Input 2)
(PWM) Pin 8	5	10	Pin 3 (Analog Input 3)
(PWM, Analog Input 7) Pin 7	6	9	Pin 4 (Analog Input 4, SCK)
(MOSI, PWM, Analog Input 6) Pin 6	7	8	Pin 5 (Analog Input 5, PWM, MISO)



أيضاً تتميز هذه المُتحكمات بالاستهلاك المنخفض جداً للطاقة. حيث يمكنها العمل على فارق جهد 1.8 فولت واستهلاك تيار يصل إلى 300 ميكروأمبير = 0.000300 أمبير (300 جزء من مليون من الأمبير) وهو ما يعني استهلاك أقل بنحو 12 مرة من استهلاك مُتحكمات megaAVR والتي تستهلك تيار كهربائي بمقدار 1 ملي أمبير (1000 ميكروأمبير) على الأقل.

إدارة استهلاك الطاقة مشروعه بالتفصيل في الفصل السابع



2.8 تمارين إضافية

- فارن بين الخصائص التي يتمتع بها المُتحكّم ATmega16 والمُتحكّم ATmega32 والمُتحكّم ATTiny84 من حيث الذاكرة وعدد أوامر البرمجة.
- كم عدد البورات التي يملكتها المُتحكّم ATTiny84 وما هي أسماؤها؟ هل جميع البورات تمتلك 8 أطراف مثل المُتحكّم ATmega16 أم يوجد اختلاف؟
- إذا قمنا بكتابة برنامج مكون من 100 أمر بلغة الأسsembli بدون استخدام أي أمر تأخير وتم تشغيل نفس البرنامج على المُتحكّم ATmega16 والمُتحكّم ATTiny84، أي المُتحكّمين سيقوم بتنفيذ البرنامج أسرع ولماذا؟
- نصيحة: لمعرفة الحل انتظ إلى فارق سرعة تنفيذ الأوامر MIPS في الصفحة الأولى من دليل البيانات لكل مُتحكّم ثم احسب زمن تشغيل البرنامج على كل مُتحكّم.
- إذا كان عدد الأوامر البرمجية لنظام تحكم يصل إلى خمسة كيلوبايت فما هو المُتحكّم المناسب لتشغيل هذا البرنامج (اختر ما يصلح من بين ATmega16, ATmega32, attiny45, attiny85, ATTiny84)؟ وإذا كان حجم البرنامج 17 كيلوبايت هل تصلح جميع اختياراتك السابقة؟



2.9 مراجع إضافية

مقارنة شاملة بين عائلة المُتحكمات ATTiny

- www.microfusion.de/e-/Microcontroller/AVR-Overview/ATtiny.html
- en.wikipedia.org/wiki/Atmel AVR_ATtiny_comparison_chart
- ATmega32-avr.com/avr-comparison/

مقارنة شاملة بين عائلة megaAVR

مرفق مع الكتاب ملف مقارنة شامل بين أفراد المُتحكمات في كلا العائلتين ATTiny و megaAVR



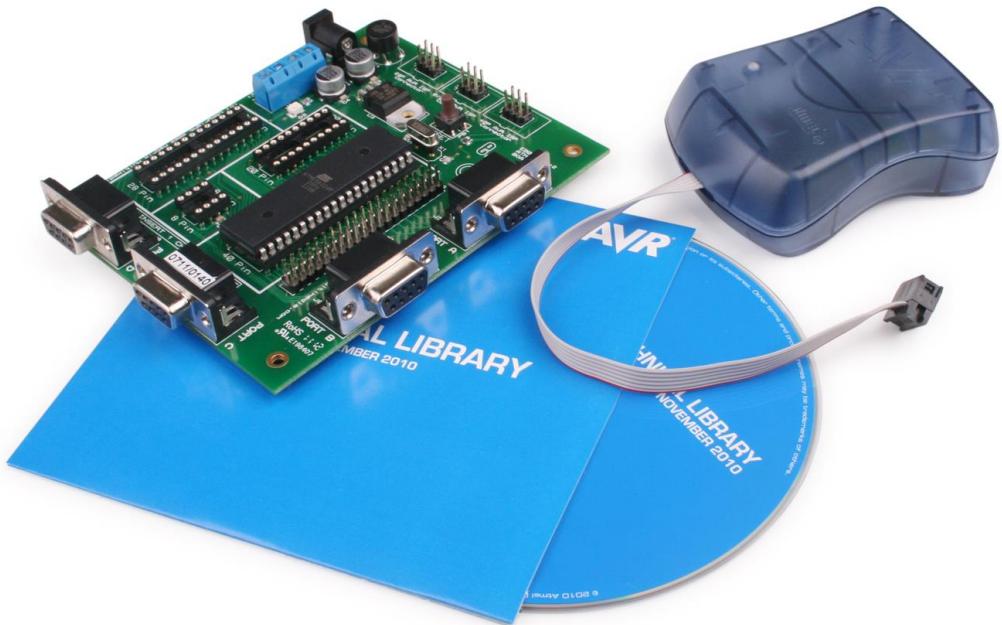
صفحة جديدة



3. تجهيز أدوات التجارب

الفصل
التي
في تطوير
المدمجة
العائد

يوضح هذا
الأدوات
التي
الأنظمة
سواء كانت
المكونات



الإلكترونية" Hardware أو الأدوات البرمجية (Softwares) ToolChain

- ✓ المبرمجات (الحرقات) (Burners).
- ✓ المكونات الإلكترونية
- ✓ البرمجيات المستخدم في التطوير Toolchain



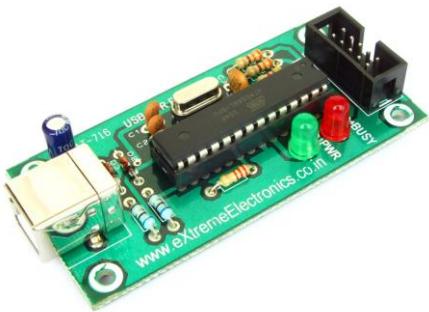
3.1 المُبرمجات

تعتبر المُبرمجات من أهم القطع الإلكترونية لتعلم المُتحكمات الدقيقة فهي المسؤولة عن تحميل البرنامج الذي نكتبه على الحاسوب إلى المُتحكم الدقيق نفسه. البعض يجب تسميته بالحرارات **Burner** نسبة إلى عملية حرق "الكود البرمجي" على المُتحكم. يمكن استخدام العديد من المُبرمجات المتوفرة في السوق ولكن هنا سأذكر لك أفضل هذه المُبرمجات وما يميزها ولكل حرية الاختيار بينها (أي واحدة منهم ستفي بالغرض).

USBasp

تعتبر أحد أبسط وأشهر المُبرمجات التي تعمل عبر مدخل **usb** التقليدي وتتميز بالسعر المنخفض حيث يمكنك صناعتها بنفسك بتكلفة 3 دولارات أو شرائها جاهزة بسعر يترواح بين 6 إلى 9 دولارات، كما أنها تحتوي على وضع الرفع البطيء **slow clock rate mode** وهو من الأوضاع المهمة في برمجة المُتحكمات عندما تعمل على سرعات منخفضة (ستحدث عن هذا الوضع بالتفصيل في فصل التلاعيب بالترددات والطاقة)، بهذا السعر المنخفض وسهولة البناء تعتبر **USBasp** أشهر مبرمجة **avr** على الإطلاق.

الصور التالية هي أشكال مختلفة من نفس المُبرمجة



الإصدار **DIP** من المُبرمجة **USBasp**



الإصدار **SMD** من المُبرمجة **USBasp**

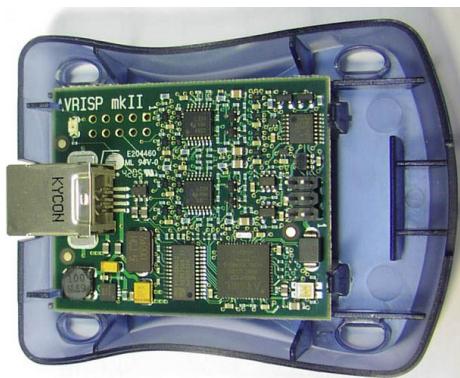
الموقع الرسمي للمُبرمجة **USBasp** يحتوي على جميع ملفات التصميم التي يمكنك تحميلها مجاناً وصناعتها بنفسك www.fischl.de/usbasp

AVRISP mkII – Atmel

المُبرمجة الرسمية من شركة أتمل، وتعد من المُبرمجات المنتظرة نسبياً حيث تدعم معظم عائلات **avr** الـ 8 بت والـ 32 بت مثل **Xmega** ويمكنها برمجة المُتحكمات بمختلف السرعات من 500 هرتز إلى 8 ميجا هرتز (هذه السرعة تمثل سرعة نقل البيانات وليس سرعة المُتحكم نفسه).

تعتبر **AVRISP mkII** أفضل من **USBasp** بكثير حيث تدعم **USBasp** سرعات برمجة بحد أقصى 5 ميجا فقط - كما أنها تدعم برمجة المُتحكمات التي تعمل بفرق جهد من 1.8 فولت حتى 5 فولت. بالتأكيد كل هذه المميزات تأتي على حساب السعر الذي يبلغ نحو 40 دولار تقريباً. يمكنك قراءة كافة التفاصيل عنها من الرابط التالي

<http://www.ATmel.com/tools/AVRISPMKII.aspx>



المُبرمجة AVRISP mkII من الداخل



المُبرمجة AVRISP mkII من الخارج

تحذير: تنتشر بعض النسخ الصيني المزورة من هذه المُبرمجة بسعر منخفض (حوالي 10 دولارات - ولديها نفس الغطاء الخارجي) وبالتالي لا تمتلك المميزات avrisp mkII احترس من هذه المُبرمجات لأنها لا تحتوي على التصميم والمكونات الحقيقة للمُبرمجة المذكورة سابقاً.

AVR Dragon

إذا كنت تريدين أقوى أداة للتعامل مع عائلات AVR إذاً عليك بهذا "الثنين" . تعتبر المُبرمجة AVR Dragon أقوى أداة للتعامل من مُتحكمات AVR فهي تعمل كمبرمجة SPI أو OCD أو PDI و يمكنها معالجة الفيوزات والبرمجة بالفولتية العالية 12 Volt – High voltage burner و يمكنها صيانة المُتحكمات المعطوبة أو نسخ و معالجة المحتوى المكتوب على المُتحكمات (سيتم شرح الفيوزات بالتفصيل في الفصل الخاص بها).

يمكنك الاطلاع على تفاصيل هذه المُبرمجة من موقع Atmel الرسمي من الرابط التالي: www.ATmel.com/webdoc/avrdragon



تتوفر هذه المُبرمجة بسعر 55 دولار أمريكي، وتعتبر من الأدوات المُخصصة للمحترفين لما تملكه من إمكانيات متقدمة.



المُبرمجات ذات التغطية العامة Universal Programmers

هذا النوع من المُبرمجات يمكنه التعامل مع جميع المُتحكمات الدقيقة وشريحة الذاكرة من مختلف الشركات فمثلاً معظم هذه المُبرمجات يمكنها برمجة PIC, AVR, ARM, 8051, EPROM والمزيد من الشريحة والمُتحكمات. غالباً تستخدمها شركات الصيانة والتطوير لأنها توفر الجهد وتتوفر شراء العديد من المُبرمجات لمختلف الأنواع.



المشكلة الوحيدة لهذا النوع هو سعرها المرتفع جداً والذي يبدأ من 100 دولار وحتى 1200 دولار (قد يبدو رقم 1200 دولار ضخم لكن لك أن تخيل أن هذه المُبرمجات يمكنها التعامل مع أكثر من 8000 شريحة إلكترونية من مختلف الشركات على هذا الكوكب).

لا أنصحك بشراء هذا النوع إلا إذا كنت ترغب في تعلم استخدام أكثر من نوع من المُتحكمات الدقيقة ولا تريد أن تتعب نفسك بشراء مُبرمجات مختلفة.



Arduino as ISP

قد يستغرب البعض لما تم وضع آردوينو في قائمة المُبرمجات، الحقيقة أن لوحات آردوينو المختلفة يمكنها العمل كمنصة تطوير AVR لأنها من الأساس عبارة عن شريحة Atmega328 مضافة إليها بعض المكونات البسيطة + محول USB-ttl converter. هذه اللوحات يمكنها برمجة AVR بطريقتين:

الطريقة الأولى: أن تقوم برفع برنامج يسمى ArduinoISP على لوحة آردوينو والذي سيقوم بتحويل اللوحة إلى مبرمج مشابه لـ USBasp ويمكن بعدها أن توصلها بابي شريحة AVR لتبرمجها كما في الصورة التالية والتي يتم استخدام لوحة آردوينو بها لبرمجة شريحة ATmega16 أو ATmega32

تستخدم لوحة آردوينو
development
الكتابة بلغة السي
برنامح آردوينو.

ANSI - C دا�ل بسهولة أن يمكنها بسهولة أن البعض إليها مضاف ما البرمجي آردوينو

للوحة اردوينو تقوم باستيراد باقي المكونات
للسراة اي مبرمجة لاملاك لوحه

استخدام آردوینو (الطريقة الأولى):

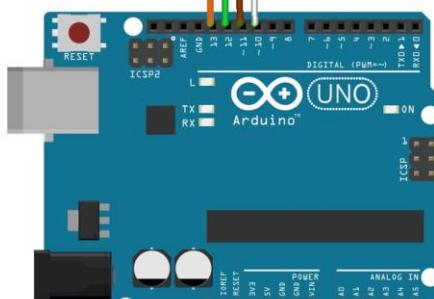
A 9V battery with the text "9V" written on it.

الطريقة الثانية: أن
AVR كنفسها board حيث يمكن أن
دالخ ANSI- C

في الواقع معيار هو إلا لغة السي المكتبات البرمجية لهذا تكتتب أي برنامج ببرنامج آردوينو.

الخبر الجيد أنه في
أردوينو فلا داعي
إضافية ويكتفى فقط أن
المطلوبة وتستغل
بهذا العمل.

الروابط التالية تشرح (مثل) كمبرمجة



- www.youtube.com/watch?v=_ZL-YNOH_jA
 - www.arduino.cc/en/Tutorial/ArduinoISP
 - www.instructables.com/id/AVR-Programming-with-Arduino-AVRdude-and-AVR-gcc/

شخصياً أستخدم المبرمج USBasp في أغلب الأوقات وتعتبر المبرمج المفضلة لدى بسبب سعرها المنخفض وسهولة صناعتها في المنزل.

في هذا الكتاب سنعتمد على شريحة ATTiny16 و شريحة AVR 328 المدعومة في الكتاب (خاصة الأكواد المذكورة في الفصل السادس حتى نهاية الكتاب) وذلك بسبب اختلاف أسماء بعض المُسجلات Register وأسماء بعض المخارج والمدخلات pins . إذا رغبت في استخدام Arduino كـ AVR board فأحرص على تغيير أسماء المُسجلات لتلبيس الأ متلة .



أيضاً سيتم شرح الـ Datasheet لكل متحكم، والتي من خلالها يمكنك تغيير هذه الأسماء بسهولة و تطبيق كافة الأكواد مع تعديلها قليلاً لتناسب مع المتحكم Atmega328 بدلاً من المتحكمات المذكورة سابقاً.

3.2 المكونات الإلكترونية

ستحتاج بعض المكونات الإلكترونية لأداء التجارب في هذا الكتاب، بعض هذه المكونات واجب توافره والبعض الآخر يستخدم فقط في التجارب الإضافية. إذا احتجت على اقتناء المكونات التي سيكتب بجانبها **(واجب)** - أما المكونات التي سيكتب بجانبها **(اختياري)** في يمكنك الاستغناء عنها (ومن ذلك أنصحك بشرائها حتى تستفيد بأكبر قدر من التجارب).

(واجب) هذا هو المتحكم الرئيسي الذي سنقوم بعمل التجارب عليه، وفي حالة عدم توافرها لديك في السوق المحلي يمكنك استخدام البديل المماثل ATmega32 والذي يماثله في معظم التركيب الداخلي باستثناء مساحة الذاكرة.

(اختياري) المتحكم ATTiny84



ATmega16
ATTiny84

(واجب) عدد 1 كريستالة (ذات طرفين) 16 ميجا



كريستالة: 16 ميجا هرتز

(واجب) عدد 2 مكثف على الأقل.

هذه المكثفات سعرها رخيص جداً وحجمها صغير وسهلة الضياع لذا يفضل أن تشتري منها 10 أو 20 قطعة (الـ 20 قطعة ستكلفك نصف دولار فقط).



مكثف سيراميكي
بيكوفاراد
22 picoFarad

(واجب) عدد 1 كابل

يُستخدم هذا الكابل في توصيل المُبرمجة بالمتحكم الدقيق (غالباً قد تجده مع المُبرمج نفسها عند لا داعي لشرائه).



كابل مبرمج ISP

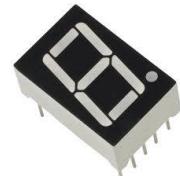


(واجب) عدد 10 قطع دايدود ضوئي (يفضل اللون الأحمر أو الأخضر)
ويفضل عدم استخدام اللون الأبيض



دايدودات ضوئية (LEDs)

(واجب) عدد 2 قطعة شريحة عرض أرقام "مقاطعة سباعية" seven Segment من المكونات الهمزة والبسطة في ذات الوقت، تستخدم في عرض الأرقام عن طريق 7 دايدودات ضوئية لذا تسمى الشريحة ذات المقاطع السبعة 7 Segment ويفضل الحصول على النوع ذا الطرف السالب المشترك common cathode



(اختياري) عدد 1 حساس حرارة والذي سنستخدمه في فصل الحساسات التماضية LM35



(اختياري) عدد 1 مقاومة ضوئية والتي سنستخدمها في فصل الحساسات التماضية



مقاومة ضوئية LDR

(واجب) عدد 1 مقاومة متغيرة ذات 3 أطراف توصيل مع عمود دوران potentiometer



(واجب) عدد 1 محرك تيار مستمر مثل الموجود في الألعاب مثل السيارات DC Motor



(اختياري) عدد 1 محرك خطوي من نوع bipolar له سلك ذا 4 أطراف ويستخدم في تقنيات التحكم للماكينات، يجب أن لا يستهلك تيار أكبر من 500 مللي أمبير (نصف أمبير) وستتعرف عليه بالقصيل في فصل المحركات.

محرك Stepper Motor (bipolar)



(اختياري) دائرة قيادة المحركات (قطرة H) وتعتبر من أهم الشرائح التي سنتستخدمها في التحكم بالمحركات سواء الـ DC أو الـ Stepper Motor

L293 H-bridge

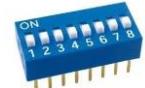


(واجب) عدد 4 مفاتيح ضغط ذات أربعة أطراف، إذا لم تكن متوفرة في السوق المحلية يمكنك شراء مفاتيح الضغط ذات الطرفين فقط.



مفتاح Push button

مفتاح DIP (اختياري) عدد 1 مصغوفة DIP – ON- OFF ذا ثمانية مفاتيح أو يمكنك شراء 2 مصغوفة ذات 4 مفاتيح switch



(واجب) عدد 8 مقاومات من كل القيم المذكورة

(10 كيلو اوم و 330 اوم) بقدرة ربع وات أو: watt 8/1

Resistors

10 كيلو اوم
330 اوم

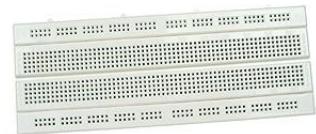


(واجب) عدد 20 سلك يمكنك شراء الأislak الخاصة بلوحة التجارب مباشرة Male to Male Jumpers أو يمكنك أن تصنعها بنفسك عبر شراء سلك (خط تليفون أرضي 0.6 مللي) وقطعها بنفسك، شخصياً أفضل استخدام الأislak الجاهزة، ويفضل أن تكون بطول ما بين 10 إلى 15 سنتيمتر.



أislak توصيل

(واجب) عدد 1 لوحة تجارب Breadboard، احرص على اقتناء لوحة كبيرة الحجم ومن النوع الجيد لأنها ستكون المنصة التي ستطبق عليها مختلف التجارب.



لوحة تجارب Breadboard



3.3 أدوات إضافية

من المحبذ أن تمتلك مجموعة من الأدوات الإضافية حيث ستساعدك كثيراً في تعلم المُتحكّمات الدقيقة والإلكترونيات بشكل عام لذا احرص على اقتتنائها إذا لم تكن لديك بالفعل. القائمة التالية هي مثال على ما ينبغي أن تمتلكه:

- جهاز قياس متعدد (Digital Multimeter (AVO meter))
- عدة اللحام (كاوية + حامل لكاوية + حامل لوحات إلكترونية (PCB holder))
- فصدير لحام من نوع 30-70
- مقص أسلاك (قصافه) + قشارة أسلاك.
- عدسة مكيرة
- مجموعة من المفكات + ماسكة أسلاك (بنس).





3.4 تجهيز البرمجيات

البرمجيات المخصصة لأنظمة المدمجة (أو كما تسمى **Firmware** فيرم-وير) يتم تصميمها باستخدام "مجموعة أدوات التطوير" أو كما تعرف باسم **Development toolchain** من الرائع أن هذه الأدوات متوفرة لمحكمات الـ AVR مجاناً وبصورة مفتوحة المصدر لجميع أنظمة التشغيل، هذا يعني أن جميع البرمجيات التي ستشتملها مجانية تماماً ويمكنك استخدامها بحرية بأي صورة سواء كانت تعليمية أو تجارية. مجموعة الأدوات هي:

المترجم AVR-GCC Compiler – البرنامج الشهير **GCC** يعتبر أشهر مترجم في العالم وهو البرنامج الرئيسي في أدوات التطوير والمسؤول عن تحويل اللغات عالية المستوى مثل **C/C++** إلى ملفات تستطيع الآلات أن تفهمها (بالتعاون مع برمجات الـ **Assembler** و الـ **Linker**) قامت شركة **ATmel** بتعديل الـ **GCC** ليعمل مع الـ **AVR** مباشرة بمختلف الإصدارات **AVR-GCC**, **AVR32-GCC**, **AVR-GCC-G++**, **AVR-GCC-F** في هذا الكتاب ستشتمل **AVR-GCC** فقط.

الأدوات المساعدة binutils – مجموعة من الأدوات التي تساعد المترجم في إتمام عملية تحويل البرنامج من اللغة عالية المستوى إلى ملف **hex** متكامل (البيكس **hex** هي صيغة نصية تمثل الـ **binary** في صورة أبسط وأكثر اختصاراً).

المكتبات البرمجية Libraries (LibC-avr) – مجموعة من الأكواد والتعرifات المكتوبة مسبقاً بلغات عالية أو أقل مستوى لتسهل عملية البرمجة على المطورين وتحتوي على بعض الأكواد والأوامر الجاهزة والتي تختصر الكثير من وقت كتابة البرامج.

المنقح GDP debugger – هذا البرنامج يستخدم في اكتشاف الأخطاء البرمجية والمساعدة في حل المشاكل التي تواجه المطور أثناء تشغيل واختبار البرنامج.

برنامج الرفع AVRdude – البرنامج المستخدم في رفع الملفات من الحاسوب إلى المُتحكمات الدقيقة من نوع AVR ويمكنه أيضاً أن يقوم بعكس هذه العملية (استخراج البرامج المكتوبة على المُتحكمات) كما يمكنه قراءة محتويات الذاكرة **EEPROM** وكتابة الفيوزات (كما سنرى في الفصل الخاص بالفيوزات).

هناك طريقتان لاستخدام هذه الأدوات، الأولى هي تحميل الـ **toolchain** ثم استخدام أي **IDE** أو حتى محرر نصوص يدعم لغة الـ **C/C++** والطريقة الثانية أن يتم تحميل منصة التطوير المتكاملة من شركة **ATmel** **Studio** – هذا البرنامج يحتوي على كل الأدوات السابقة مدمجة بداخله (باستثناء **AVRdude** فقط).

الطريقة الأولى تصلح لجميع أنظمة التشغيل ومناسبة جداً لأنظمة **Linux** و **Mac** أما الثانية مع الأسف برنامج **ATmel Studio** متوفّر على نظام **Windows** فقط، على أي حال سأقوم بشرح كلا الطريقتين حتى يصبح لك حرية اختيار البرنامج وحرية استخدام أي نظام تشغيل تريده.

أيضاً يمكنك استخدام برنامج المحاكاة الشهير **Protues** في محاكاة جميع الأمثلة المذكورة في فصول الكتاب دون الحاجة لشراء مكونات إلكترونية حقيقة، مع العلم أن برنامج **Protues** يمكنه العمل على نظام **Linux** أيضاً عبر استخدام محاكي **Wine HQ** و **Windows**.

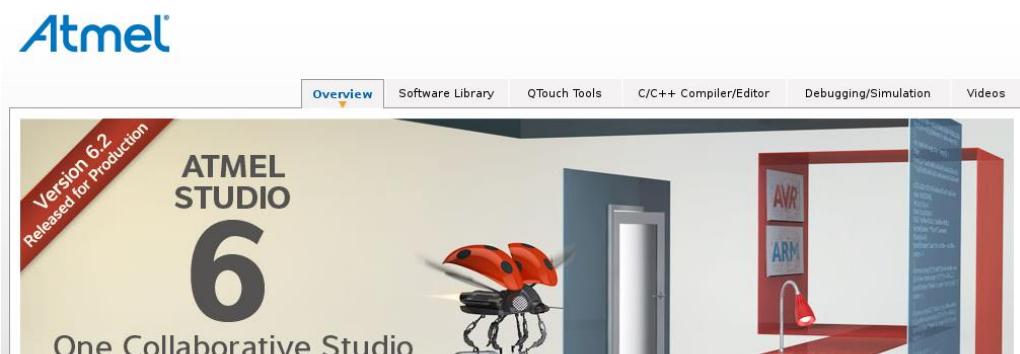
معلومات إضافية: برنامج **Atmel Studio** يعمل بنفس الـ **toolchain** المذكورة سابقاً و منها المترجم **AVR-GCC** كذلك نجد أن برنامج **Arduino** يعمل بنفس الـ **toolchain**، وهذا يعني أنه يمكنك استخدام برنامج **Arduino** كـ **IDE** خفيفة لكتاب برامج بلغة الـ **C/C++** لبرمجة الـ **AVR** ولكنني لا أجد هذا الخيار لأنه يفتقر للكثير من الوظائف الاحترافية والهامة لمبرمجي الأنظمة المدمجة.



تجهيز الأدوات على نظام ويندوز

توجه إلى موقع شركة Atmel لتحميل برنامج Atmel Studio من الرابط التالي

<http://www.ATmel.com/tools/atmelstudio.aspx>



البرنامج
والمضاف إليها
الـ .NET. (هذه
على كل
المطلوبة)
بحجمها الكبير
على أن يكون
سريع
تحميل

الموقع أن
جديد (أو
إذا كان لديك
بالفعل،
مجاني تماماً
تحميل جميع
بعد
مجاني.
تسجيل
تحميل

من تنصيب

اختر نسخة
المتكاملة
باقية أدوات
النسخة تحتوي
الملفات
وعادة ما تتميز
لذا احرص
لديك اتصال
بإنترنت عند
البرنامج.
سيطلب منك
تسجيل حساب
تسجيل دخول
حساب
الحساب
وسيوفر لك
أدوات الشركة
الانتهاء من
الحساب يمكنك
البرنامج.

بعد الانتهاء من تنصيب Atmel studio سنقوم بتحميل برنامج AVRdudess وهو برنامج مضاف إليه واجهة رسومية رائعة ومزودة بالعديد من الخيارات التي تسهل برمجة شرائح AVR، يمكنك تحميل البرنامج من الموقع التالي:

<http://blog.zakkemble.co.uk/avrdudess-a-gui-for-avrdude>



AVRDUDESS – A GUI for AVRDUDE

17
2013

Arduino, AVR, Microcontroller

by Zak Kemble

تصيب

AVRDUDESS is a GUI for AVRDUDE, a tool for programming Atmel microcontrollers.

Some key features:

- Supports all programmers and MCUs that AVRDUDE supports
- Supports presets, allowing you to change between devices and configurations quickly and easily
- Drag and drop files for easy uploading
- Automatically lists available COM ports
- Cross-platform with the use of Mono for Linux & Mac OS X

Downloads



تعريفات المبرمج USBasp

هذا الجزء قد يختلف على حسب المبرمجة التي ستستخدمها، شخصياً أستخدم USBasp لأنها؛ كما ذكرت سابقاً رخيصة ومفتوحة المصدر ويمكنك صناعتها بنفسك. لذا اخترتها كأداة رئيسية للبرمجة في هذا الكتاب.

ملاحظة: إذا كنت تستخدم أحد مبرمجات Atmel مثل AVRISP أو AVR Dragon يمكنك تحميل التعريفات الخاصة بها من نفس صفحة برنامج studio

في البداية توجه إلى موقع USBasp الرسمي وقم بتحميل الملف المضغوط الذي يحتوي على جميع ملفات المشروع (ملفات التصميم والتعريفات)

<http://www.fischl.de/usbasp>

USBasp - USB programmer for Atmel AVR controllers

USBasp is a USB in-circuit programmer for Atmel AVR controllers. It simply consists of an ATMega88 or an ATMega8 and a couple of passive components. The programmer uses a firmware-only USB driver, no special USB controller is needed.



Features

- Works under multiple platforms. Linux, Mac OS X and Windows are tested.
- No special controllers or smd components are needed.
- Programming speed is up to 5kBytes/sec.
- SCK option to support targets with low clock speed (< 1,5MHz).
- Planned: serial interface to target (e.g. for debugging).

Download

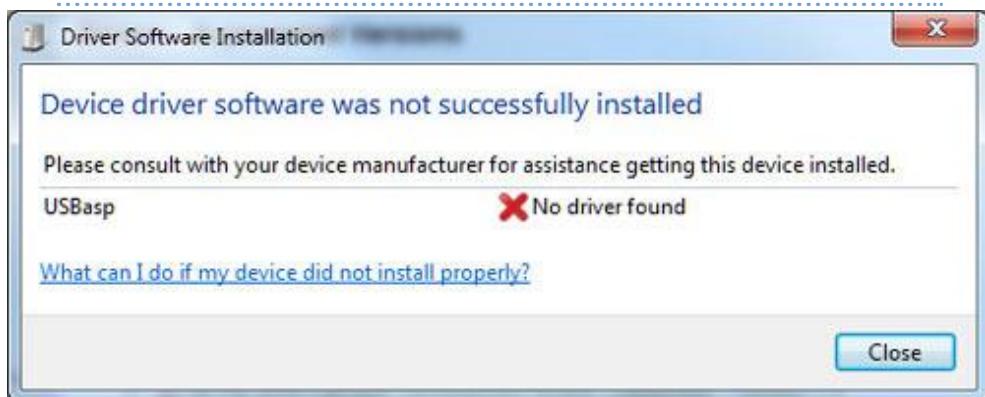
Firmware and circuit

The following packages include circuit and firmware.

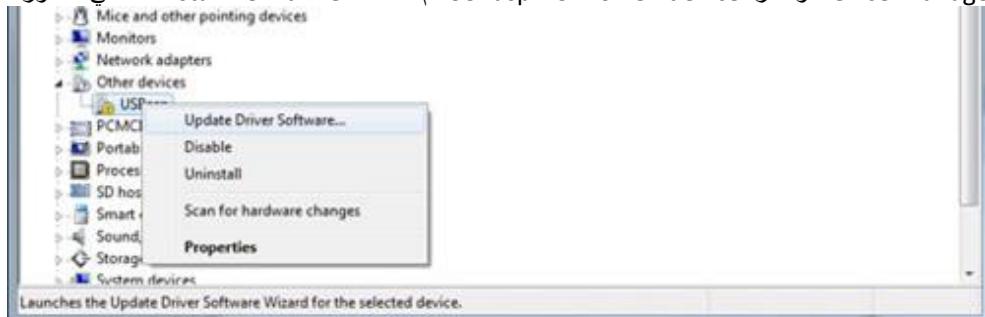
[usbasp.2011-05-28.tar.gz](#) (519 kB) TPI support (upcoming release of avrdude will use it), supports programmers with ATMega88 and ATMega8.
[usbasp.2009-02-28.tar.gz](#) (260 kB)
[usbasp.2007-10-23.tar.gz](#) (172 kB)



بعد الانتهاء من التحميل قم بتوصيل لوحة USBasp بالحاسوب وقم بفك ضغط الملف، لاحظ أنه بمجرد توصيل USBasp سيخبرك نظام ويندوز بأنه لم يستطع أن ينصب التعريفات كما في الصورة التالية:



افتح مدير الأجهزة Device Manager واختر USBasp - Unknown device ثم اضغط install new driver كما في الصورة التالية:



اختر مكان ملفات التعريفات (الموجودة داخل الملف الذي قمنا بتحميله سابقاً)



إذا ظهرت رسالة تطلب تأكيد تنصيب التعريفات فم بالضغط على Install



رائع ! أنت الآن جاهز لتبدأ العمل على صناعة الأنظمة المدمجة بالـ AVR ():

إذا كنت تستخدم نظام windows 8.1 أو 10 يجب أن تقوم بإغلاق (نظام التعريفات الموقته)، يمكنك قراءة التفاصيل من الرابط التالي

http://www.atadiat.com/usbasp_win_driver/

<https://openchrysalis.wordpress.com/2014/09/26/installing-usbasp-driver-software-in-windows-8-1/>

تجهيز الأدوات على أنظمة لينكس

في أنظمة لينكس \ ماك FreeBSD يمكنك تنزيل جميع برامج الـ toolchain من مستودعات البرامج الرسمية لكل نظام تشغيل، كما ستحتاج لملف MakeFile والذي سيقوم بتحويل الكود إلى ملف hex بصورة تلقائية (ملف الـ MakeFile مرفق مع الكتاب).

أيضاً ستحتاج IDE أو محرر نصوص يدعم البرمجة مثل البرنامج الرائع CodeBlocks (والذي يستخدمه شخصياً) أو Sublime أو Eclipse أو المحرر النصي Emacs أو Vim أو Geany.

Ubuntu / Debian نظام

يمكنك تنصيب جميع الأدوات مباشرة عبر الأمر التالي (تكتب في طرفية سطر الأوامر Terminal)

`sudo apt-get install gcc-avr binutils-avr gdb-avr avr-libc avrdude`

Fedora / RedHat / CentOS نظام

يمكنك استخدام برنامج yum أو DNF (الموجود في نظام فيدورا لينكس 22 أو أعلى) وذلك عبر الأوامر التالية

`sudo yum install avr-gcc avr-binutils avr-libc avr-gdb avrdude`

لنظام فيدورا 22 أو أعلى

`sudo dnf install avr-gcc avr-binutils avr-libc avr-gdb avrdude`



تثبيت تعريفات **USBasp** على لينكس

يحتوي الملف الرسمي على تعريفات نظام لينكس (متوافقة مع جميع الأنظمة بلا استثناء) وهي عبارة عن ملف `udev rules` وسكريبت تثبيت، كل ما عليك فعله أن تفتح المجلد الذي يحتوي على التعريفات وتشغل سكريبت التثبيت بصلاحية الرووت كما في الصورة التالية:

```
@localhost ~]$ cd "Downloads/usbasp.2011-05-28/bin/linux-nonroot"
@localhost linux-nonroot]$ ls
99-USBasp.rules install_rule
@localhost linux-nonroot]$ sudo ./install_rule
```

3.5 مراجع إضافية

- <http://www.ladyada.net/learn/avr/programmers.html>
- <http://avrprogrammers.com/programmers/all>
- <http://www.instructables.com/id/AVR-ISP-programmer>
- <http://www.instructables.com/id/Turn-Your-Arduino-Into-an-ISP>
- http://elm-chan.org/works/avrx/report_e.html
- <http://www.instructables.com/id/Programming-an-Atmel-AtTiny85-using-Arduino-IDE-an/>



.....صفحة جديدة



4. أساسيات التحكم GPIO Basics

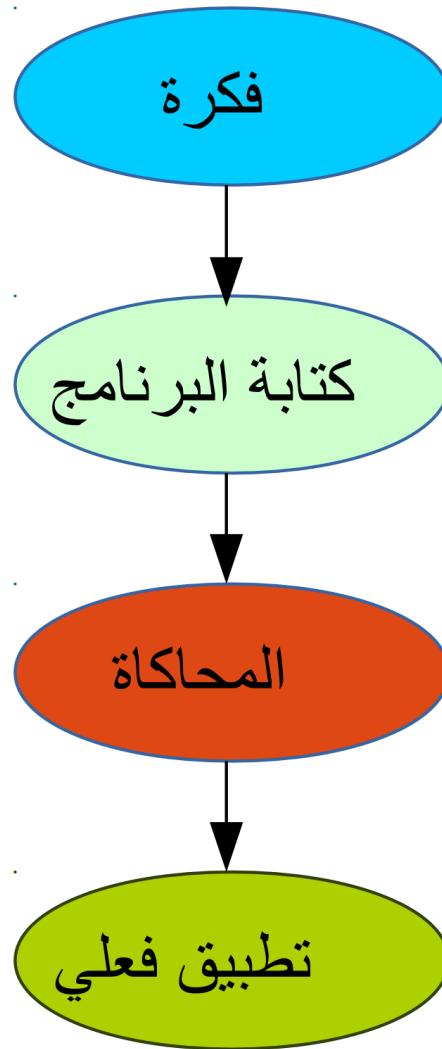


في هذا الفصل سنتعلم أساسيات تشغيل أطراف المُتحكم الدقيق وتشغيل المنافذ لعمل كدخل أو كخرج. كما سنقوم بمجموعة من التجارب لتشغيل بعض العناصر الإلكترونية البسيطة مثل LEDs, Switchs, 7-Segments..الخ.

- Hello World: المثال الأول: ✓
- أساسيات برمجة أطراف AVR: ✓
- المثال الثاني: تشغيل مجموعة دايوادات ضوئية ✓
- المثال الثالث: تشغيل جميع أطراف الپورت A والپورت B ✓
- المثال الرابع: تشغيل المقاطعة السباعية Segment7 ✓
- قراءة الدخل الرقمي: ✓
- Internal & External Pull-Up: ✓
- المثال السادس: قراءة أكثر من مفتاح ✓
- مفهوم الـ Bouncing وطرق الـ De-Bouncing: ✓



في جميع التجارب التالية سنتبع أسلوب التصميم **Design** ثم المحاكاة **Simulate** ثم التنفيذ على لوحة التجارب **Prototype** وذلك لتسهيل تعلم البرمجة، مع ملاحظة أنه هناك بعض الأمثلة التي قد لا تصلح للمحاكاة ويجب أن تنفذ مباشرة على لوحة التجارب كما سنرى في الفصول المتقدمة (مثل الـ **fuses** وإدارة الطاقة).



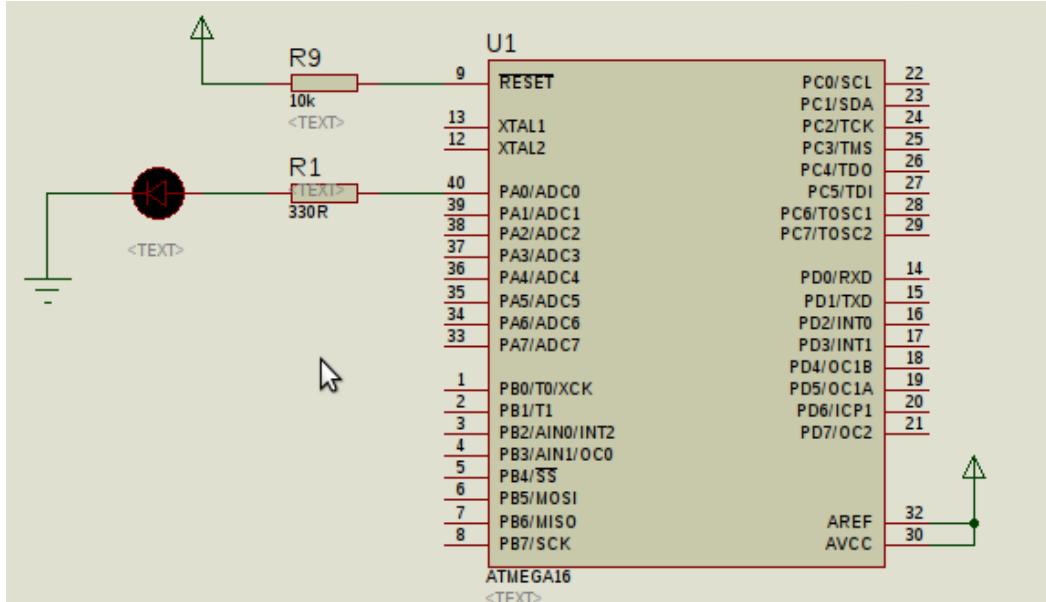
ملاحظة: جميع التجارب على لوحة الاختبارات **Breadboard** (التطبيق الفعلي) + العديد من التجارب الاضافية ستشرح على هيئة فيديوهات مستقلة عن الكتاب



Hello World المثال الأول: 4.1

يعتبر تشغيل دايمود ضوئي وإطفاؤه لمدة زمنية معينة Blinking Led هو المثال الأشهر لبدء أي عملية تطوير في عالم الأنظمة المدمجة. في هذا المثال سنتعرف على أساسيات التحكم في أطراف الـ AVR microcontrollers وتشغيلها كـ GPIO (منفذ إدخال وإخراج عامة).

سنستخدم في هذا المثال دايمود ضوئي Led يتم توصيله على الطرف PA0 ويمكنك استخدام إما ATtiny84 أو ATmega16 (كلاهما يمتلكان الطرف PA0) كما هو موضح في المخطط التالي:



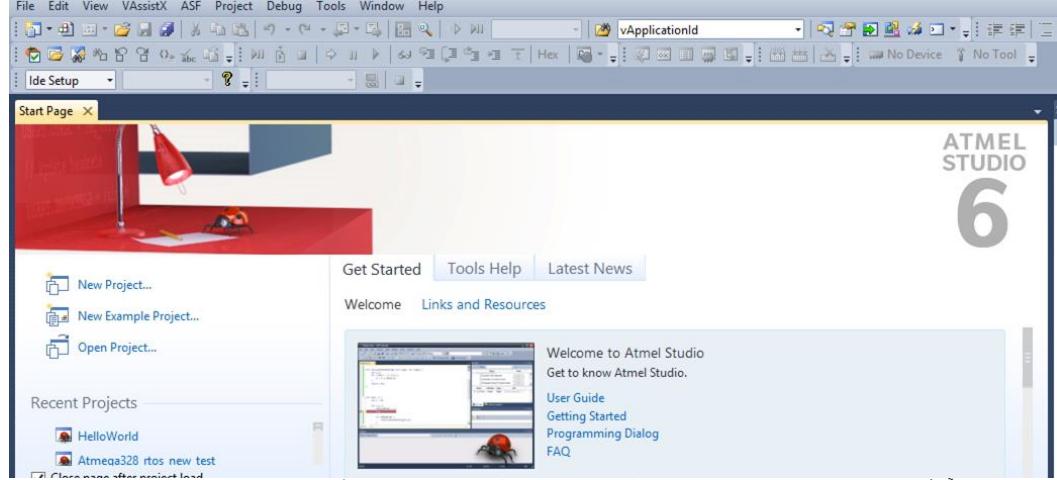
ملاحظة: كلمة مُخطط Schematic تعني الرسمة التي تشرح توصيل المكونات الإلكترونية ببعضها البعض، دائمًا ما يتم استخدام المخططات لشرح أي تصميم إلكتروني سواء كان بسيطًا أو مُعقّدًا. في هذا الكتاب سأستخدم برنامج بروتس لرسم معظم المخططات للدوائر التي سنقوم بتجربتها على مدار الأمثلة القادمة.



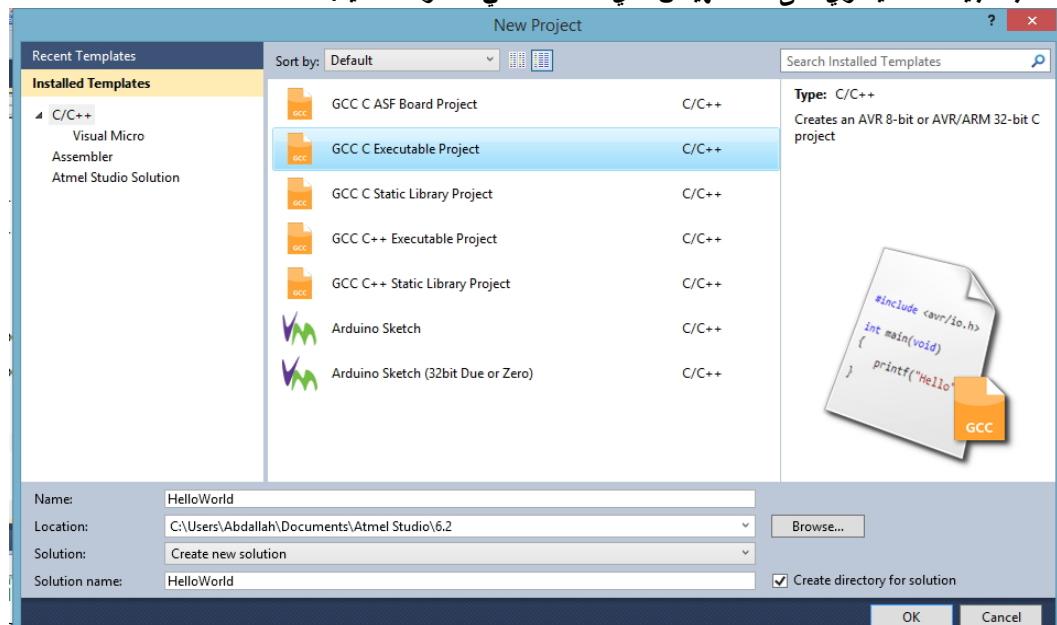
كتابة البرنامج على Atmel Studio

قم بتشغيل برنامج Atmel studio ثم اختر New Project ثم اختر

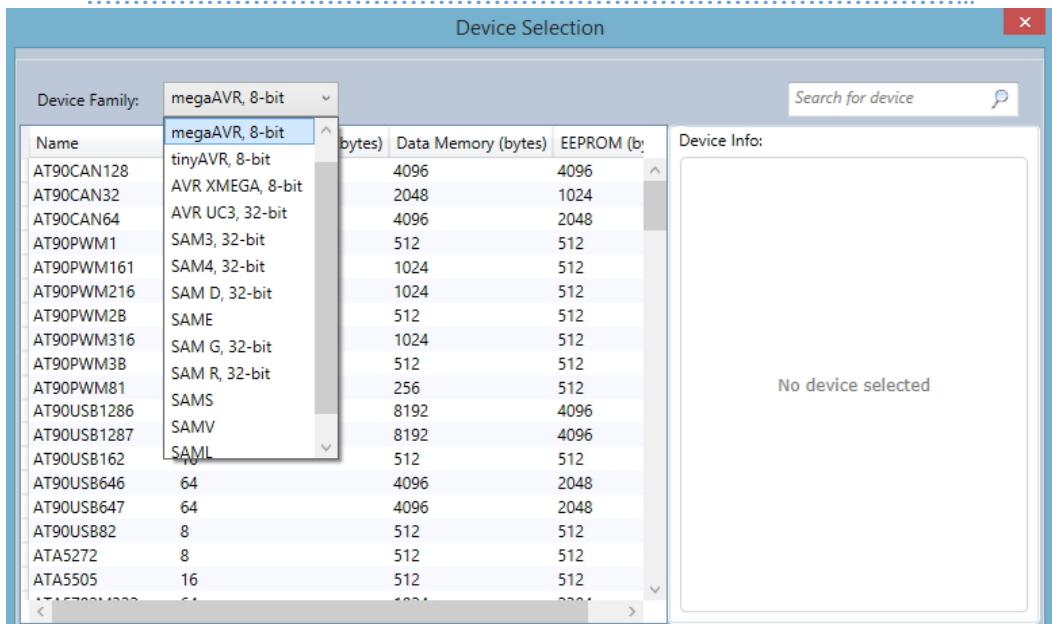
من الصفحة التي ستظهر اختر مشروع جديد بلغة السي **GCC – C- Executable** وقم باختيار اسم المشروع والمجلد الذي سيحفظ به المشروع من الشريط



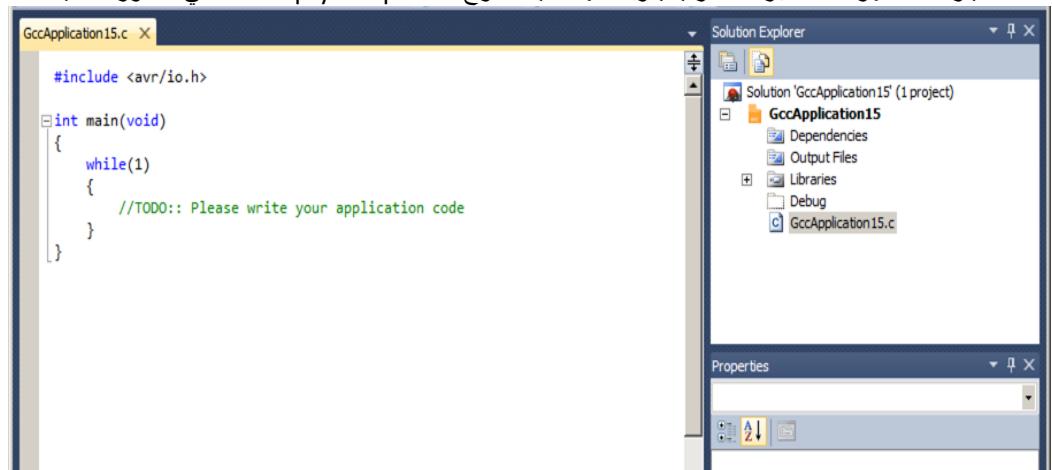
السفلي. تذكر هذا المجلد جيداً لأنه سيحتوي على ملف الهيكس الذي سنتعمله في الخطوات التالية.



الآن يمكنك اختيار عائلة ونوع المتحكم الدقيق المستخدم، كما يمكنك استخدام مربع البحث على الجانب الأيمن من الصفحة (اختر المتحكم ATmega16 أو ATtiny84).



بعد الانتهاء من هذه الاختيارات ستظهر شاشة البرمجة الرئيسية بداخلها "هيكل فارغ" Empty template كما في الصورة التالية



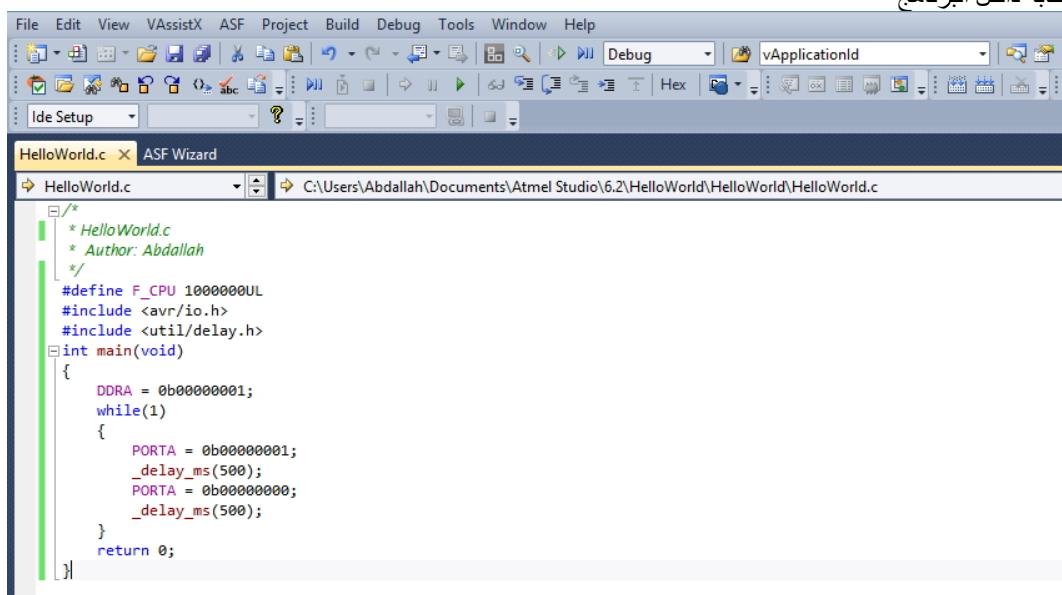


والآن قم بكتابه أول برنامج

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <avr/delay.h>

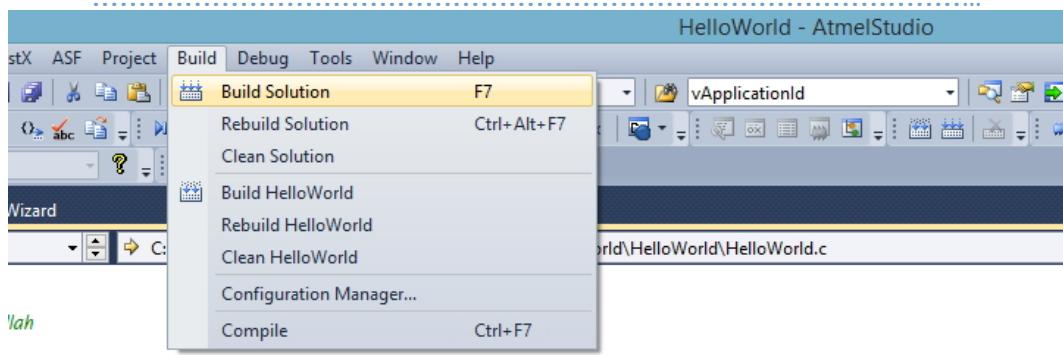
int main(void)
{
    DDRA = 0b00000001;
    while(1)
    {
        PORTA = 0b00000001;
        _delay_ms(500);
        PORTA = 0b00000000;
        _delay_ms(500);
    }
    return 0;
}
```

شكل الكود بعد كتابة داخل البرنامج

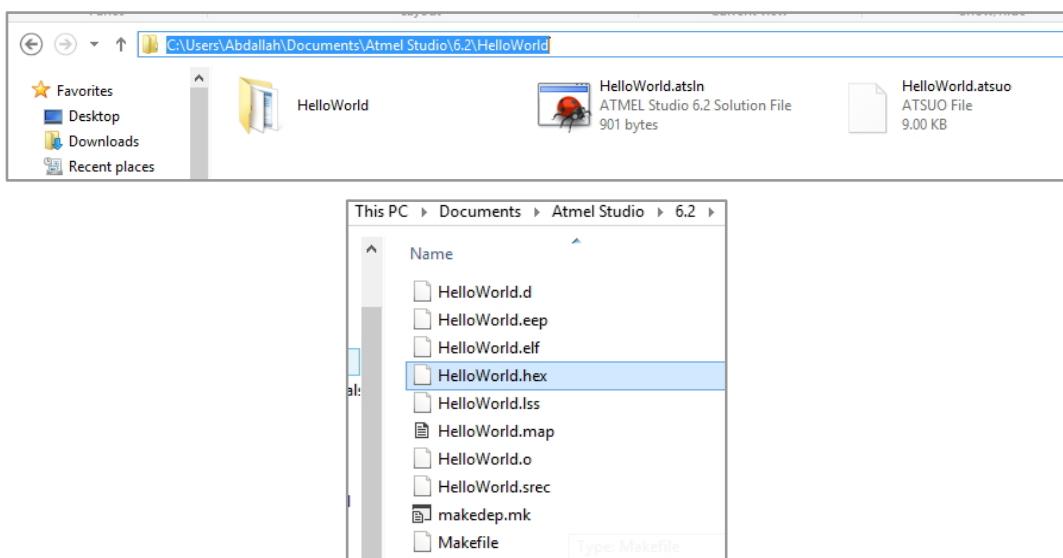


ترجمة الكود

لترجمة البرنامج وتحويلة من لغة السي إلى ملف الهيكس يمكنك الضغط على زر F7 أو اختيار "بناء البرنامج" من قائمة Build → Build Solution



بعد الانتهاء من ترجمة البرنامج ستتجد ملف الهيكس في المجلد الذي اخترناه في الخطوة الأولى، الآن يمكننا البدء في محاكاة التجربة على برنامج بروتوس (Breadboard) أو رفع ملف الهيكس على المتحكم الدقيق مباشرةً (على الأ



يمكنك استخدام طريقة تحويل الملفات باستخدام المترجم GCC مباشرةً دون استخدام برنامج Atmel studio مثل ما هو موضح في ملحق المراجع **makefile**

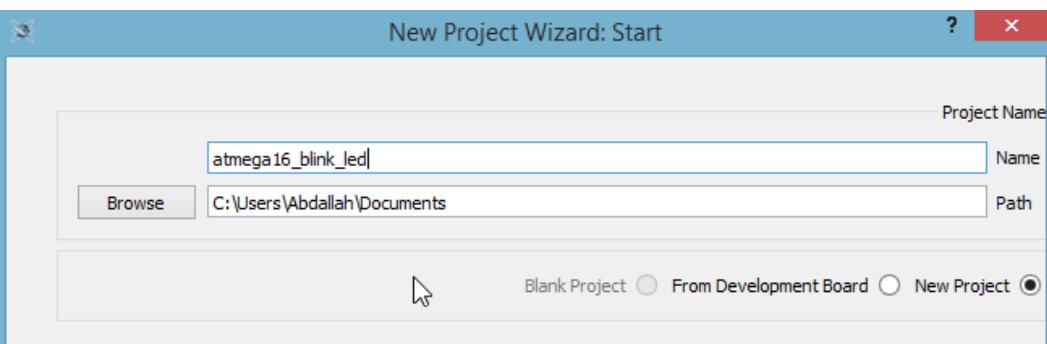
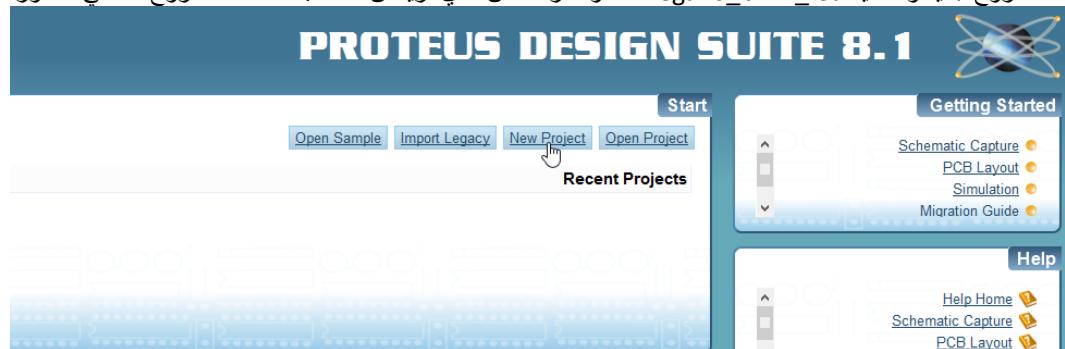


محاكاة التجربة على برنامج بروتس

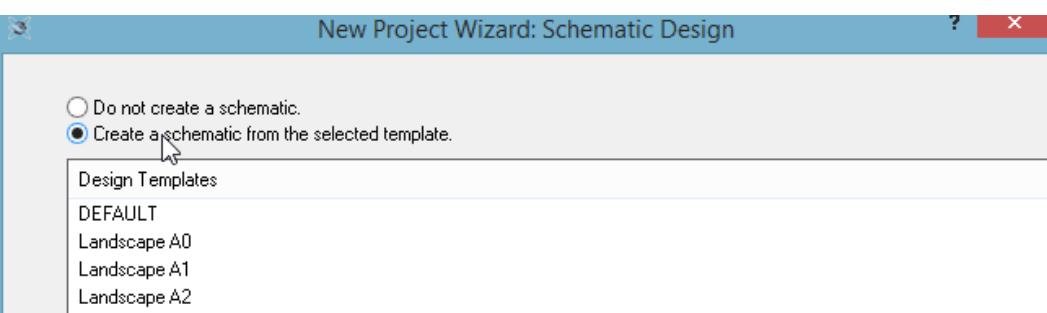
والآن سنقوم بمحاكاة التجربة على برنامج بروتس الشهير والذي يعد أفضل برنامج محاكاة في مجال الإلكترونيات خاصة مع المُتحكمات الدقيقة. البرنامج مخصوص للعمل على نظام تشغيل ويندوز ومع ذلك يمكنك تشغيله على أنظمة لينكس بسهولة باستخدام محاكي برمجي ويندوز **Wine** (وهو ما أفعله شخصيا لأنني أفضل استخدام نظام لينكس).

يمكنك استخدام أي إصدارة من برنامج بروتس سواء كانت 7.8 SP2 أو الإصدار 8.1 مع العلم أن جميع الملفات المرفقة مع الكتاب تم تصميمها واختبارها على كلا الإصدارين

في البداية قم بعمل مشروع جديد ولنسميه **ATmega16_blink_led** واختر المكان الذي تريده أن تحفظ به ملفات المشروع كما في الصور التالية:



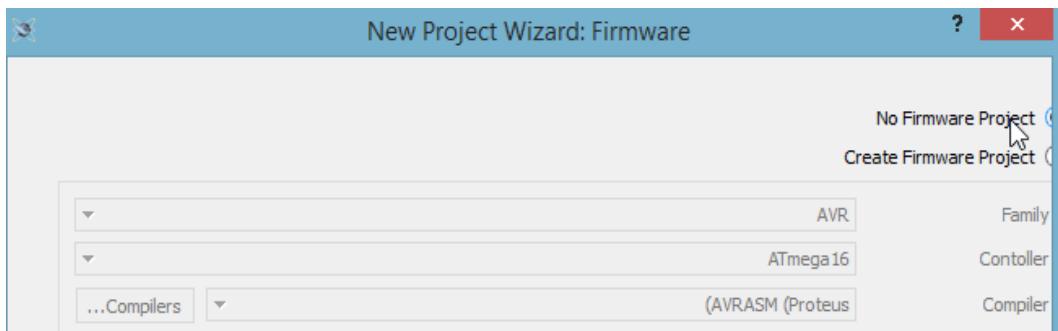
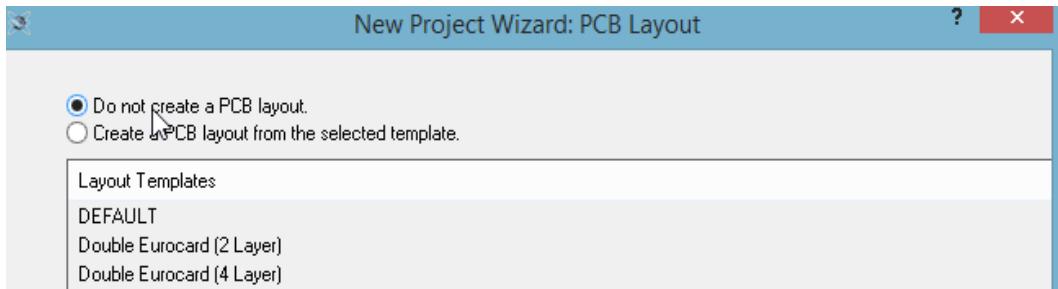
اختر تصميم مخطط جديد Create a new schematic بالمقاييس الافتراضية Default



لن نحتاج أن نصنع تصميم PCB في الوقت الحالي، لذا قم باختيار **Don't Create PCB** ثم اختر في الصفحة التي تليها **No Firmware** ملاحظة: الإصدارات الخاصة ببرنامج بروتس بدء من 8 أو أعلى تدعم برمجة المُتحكمات من داخل البرنامج باستخدام المترجم **AVRASM** أو **gcc-avr**



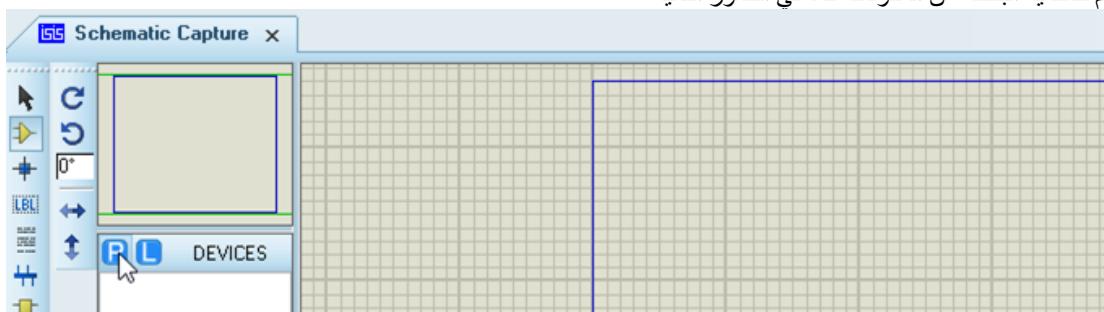
لكتنا لن نستخدم هذه الخاصية الأن وسنكتفي باستخدام برنامج Atmel studio أو CodeBlocks كبيئة برمجة مع gcc-avr



وأن أصبح لدينا ملف المخطط Schematic جاهز لنبدأ بتوصيل المكونات الإلكترونية مع بعضها البعض، في التجارب القادمة سنقوم باستخدام المكونات التالية (سنقوم بإضافتها في قائمة المكونات المستخدمة في المحاكاة من خلال الضغط على زر **P** من قائمة **devices**)

- ATmega16
- LED (yellow)
- Resistor 330
- LED bar

يمكنك استخدام خاصية البحث عن المكونات كما في الصور التالية





Pick Devices

Keywords: atmega16

Match Whole Words?

Show only parts with models?

Category: (All Categories) Microprocessor ICs

Results (11):

Device	Library	Description
ATMEGA16	AVR2	16 Kbytes Flash, 1088 Bytes SRAM, 512 Bytes EEPROM, ADC, Analog Comparator, TWI, SPI, 4 Ports, 3 Timers, 1 USAF
ATMEGA16	Disk Library	AVR2 LIB
ATMEGA16	Created On	26-٢-١١، نوافم، ٠٥:٢٩:٤٥
ATMEGA16	Category	Microprocessor ICs
ATMEGA16	Sub-category	Atmel
ATMEGA16	Manufacturer	Atmel
ATMEGA16	Description	16 Kbytes Flash, 1088 Bytes SRAM, 512 Bytes EEPROM, ADC, Analog Comparator, TWI, SPI, 4 Ports, 3 Timers, 1 USAF
ATMEGA16	1 USAF	
ATMEGA16P-32PIN-AV16	AVR2	16 Kbytes Flash, 1248 Bytes SRAM, 512 Bytes EEPROM, ADC, Analog Comparator, TWI, SPI, 3 Ports, 3 Timers, 1 USAF
ATMEGA169	AVR2	16 Kbytes Flash, 1248 Bytes SRAM, 512 Bytes EEPROM, ADC, Analog Comparator, USI, SPI, 7 Ports, 3 Timers, 1 USAF
ATMEGA169P	AVR2	16 Kbytes Flash, 1248 Bytes SRAM, 512 Bytes EEPROM, ADC, Analog Comparator, USI, SPI, 7 Ports, 3 Timers, 1 USAF

ATMEGA16 Preview.

VSM DLL Model [AVR2.DLL]

PCB Preview.

ابحث عن جميع المكونات المذكورة بالأعلى ثم ضفها إلى القائمة

Pick Devices

Keywords: led

Match Whole Words?

Show only parts with models?

Category: (All Categories) Analog ICs Diodes Electronic-mechanical Inductors Microprocessor ICs Modeling Primitives Operational Amplifiers

LED-YELLOW

Results (26):

Device	Library	Description
DIODE-LED	DEVICE	Generic light emitting diode (LED)
HD12864-4	DISPLAY	128x64 Graphical LCD with SED1565 controller, Parallel data input, LED Backlight
HD12864L-6	DISPLAY	128x64 Graphical LCD with SED1565 controller, Selectable Interface, LED Backlight
HDM32GS12-B	DISPLAY	122x32 Graphical LCD with SED1520 controllers, LED Backlight
HDM32GS12Y-3	DISPLAY	122x32 Graphical LCD with SED1520 controllers, Selectable Interface, VAC LED Backlight
LED	DEVICE	Generic light emitting diode (LED)
LED-BARGRAPH-GRN	DISPLAY	Green LED Bargraph Display
LED-BARGRAPH-RED	DISPLAY	Red LED Bargraph Display
LED-BIBY	ACTIVE	Animated Bi-Colour LED model (Blue/Yellow) with Self-flashing
LED-BIGY	ACTIVE	Animated Bi-Colour LED model (Green/Amber) with Self-flashing
LED-BIRG	ACTIVE	Animated Bi-Colour LED model (Red/Green) with Self-flashing
LED-BIRY	ACTIVE	Animated Bi-Colour LED model (Red/Yellow) with Self-flashing
LED-BLUE	ACTIVE	Animated LED model (Blue)
LED-GREEN	ACTIVE	Animated LED model (Green)
LED-RED	ACTIVE	Animated LED model (Red)
LED-YELLOW	ACTIVE	Animated LED model (Yellow)
LUMILED	DISPLAY	LED-YELLOW
MATRIX-8x7	Disk Library	ACTIVE LIB
MATRIX-8x7	Created On	26-٢-١١، نوافم، ٠٥:١٤:٢٨
MATRIX-8x7	Category	Optoelectronics
MATRIX-8x7	Description	Display

LED-YELLOW Preview.

Schematic Model [LEDA]

PCB Preview.

No PCB Preview.

بعد إضافة جميع المكونات سنجد أن المستطيل الجانبي الخاص بالمكونات أصبح يحتوي على معظم المكونات التي تحتاجها للتجارب القادمة كما في الصورة التالية:

Schematic Capture

ATMEGA16

DEVICE

ATMEGA16

CERAMIC22P

CRYSTAL

LED-BARGRAPH-RED

LED-YELLOW

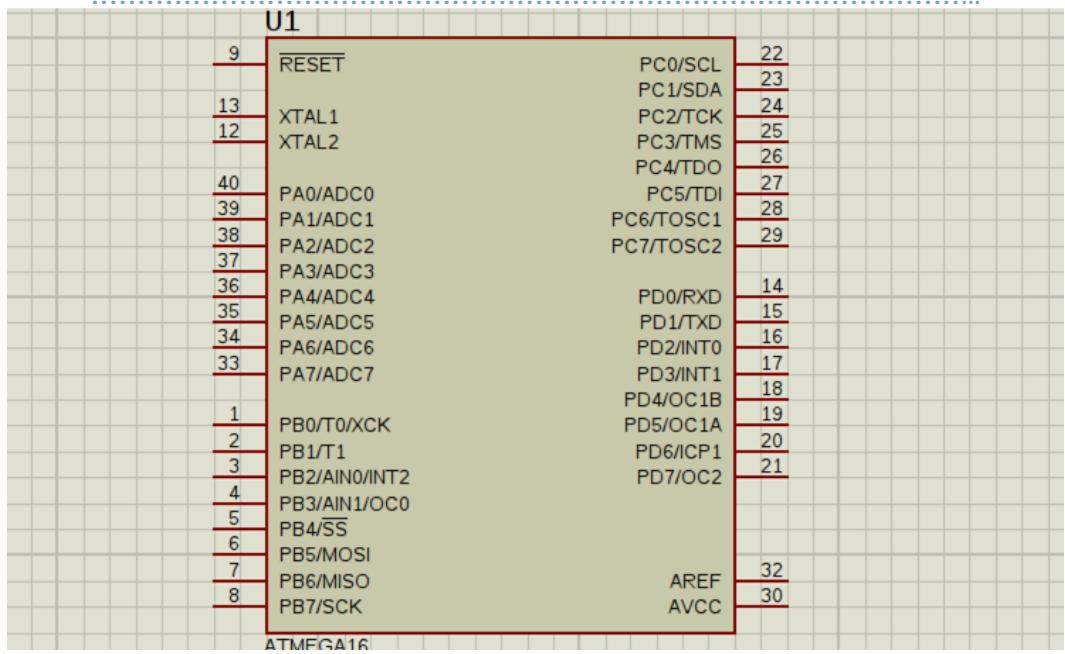
MINRES330R

والآن سنبدأ في بناء أول دائرة لتجربة الـ Blinking led، في البداية سنقوم بوضع المتحكم ATmega16 داخل إطار رسم برنامج بروتوكس كما في الصورة التالية:



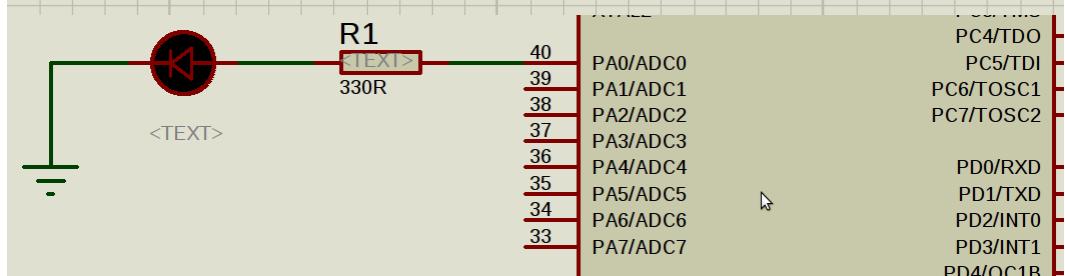
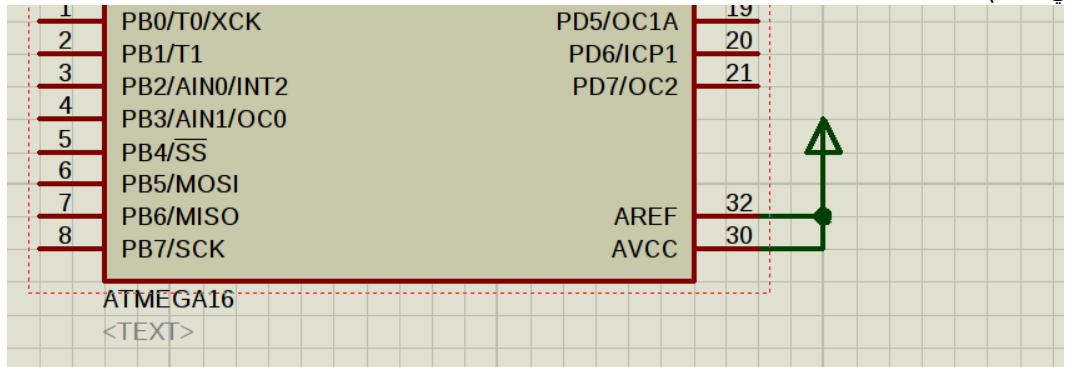
قائمة
من برنامج
(القائمة التي
رموز
الموجب
ومنها
power
بتوصيلها
AREF و
(ستحدث
المخرجين
الفصل
بالمحول

ثم سخنار
terminal
بروتس
تحتوي على
- البطارية
والسالب)
سنضيف
ونقوم
بالمدخل
AVCC
عن كل
بالتوصيل في
الخاص

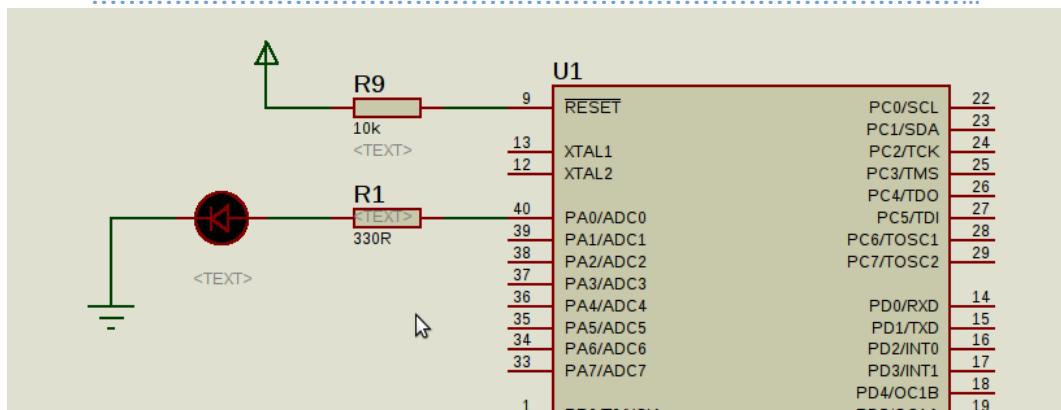


التناطري (ADC رقمي).

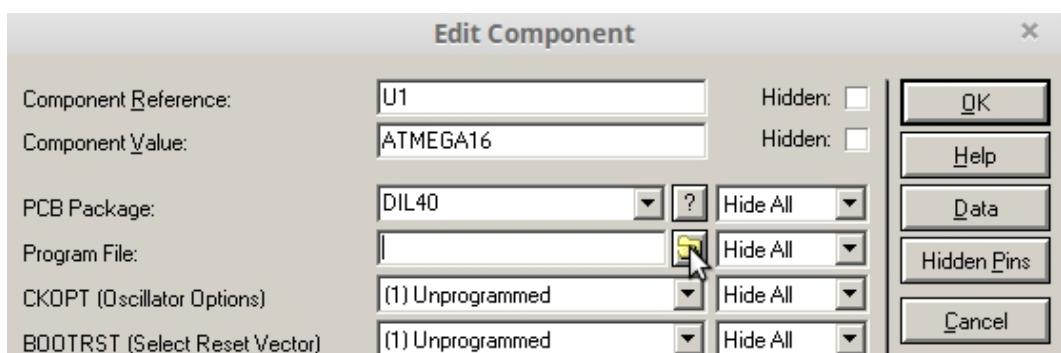
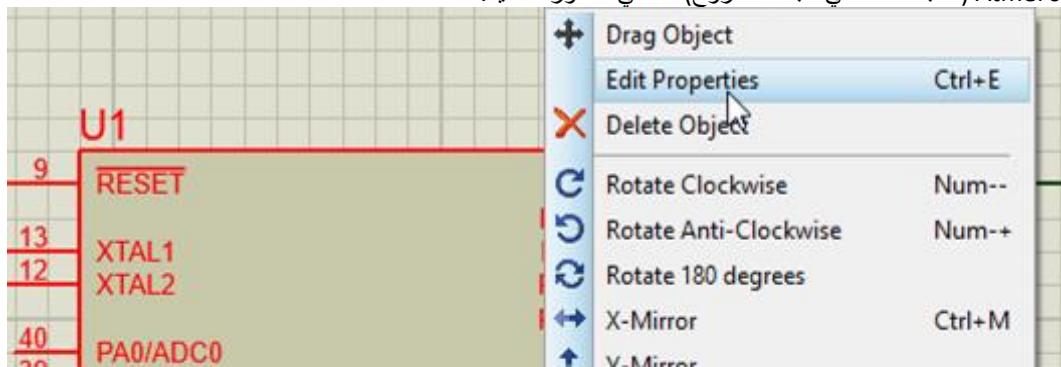
و الآن قم
الدايود
والمقاومة
ثم PA0
الطرف
على
Ground
إضافتها من
Terminals
في الصورة

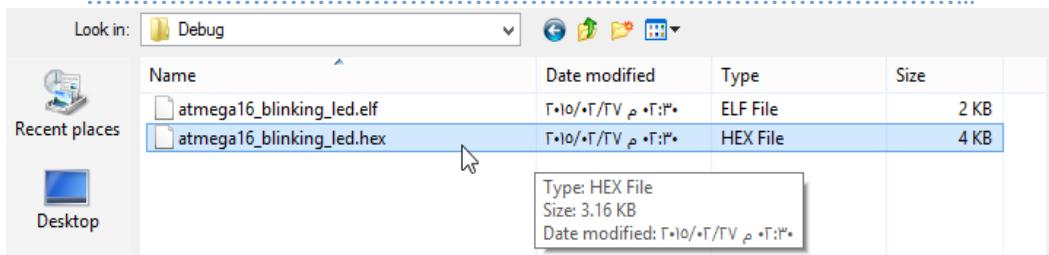


قم بإضافة مقاومة أخرى ووصلها بالمدخل RESET في الطرف الأيسر من المتحكم ثم وصل الطرف الآخر بعلامة Power - وقم بتعديل قيمة المقاومة لتتصبح 10 كيلو (تكتب k10) كما في الصورة التالية:



بذلك تكون قد انتهينا من توصيل المكونات الأساسية ويتبقى فقط إضافة ملف الهيكس hex الخاص بالكود الذي كتبناه على برنامج CodeBlocks وذلك عبر الضغط بالزر الأيمن على المتحكم ATmega16 واختيار "تعديل خصائص المتحكم" ثم الضغط على Program File واختيار ملف الهيكس الذي صنعناه باستخدام Atmel studio (ستجد الملف في مجلد المشروع) كما في الصورة التالية:



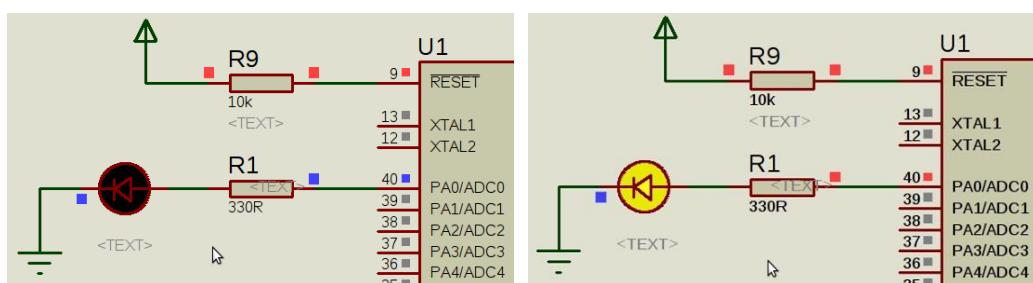


وأخيراً قم بتعديل CKSEL Fuses باختيار Int. RC 1 MHz كما في الصورة التالية



قم بحفظ
 عبر الضغط
 ثم قم بتشغيل
 الضغط على
 الشريط
 ل البرنامج بروتوكول

والآن يفترض





4.2 شرح المثال الأول وأساسيات برمجةavr

عادة ما تم تقسيم البرامج البسيطة للمتحكمات إلى 3 أجزاء أساسية:

- استدعاء المكتبات وتعريف الثوابت
- الدالة الرئيسية للبرنامج Main Function
- الدوال الإضافية "إن وجدت"

هيكل البرامج

الشكل التالي يوضح الهيكل الرئيسي لمعظم البرامج الخاصة بمتتحكمات AVR

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <avr/delay.h>

int main(void) {
    DDRA = 0b00000001;
    while(1) {
        PORTA = 0b00000001;
        _delay_ms(500);
        PORTA = 0b00000000;
        _delay_ms(500);
    }
    return 0;
}
```

استدعاء المكتبات والتعريفات

البرنامج الرئيسي

في الجزء الأول من البرنامج نجد الأمر `#define F_CPU 1000000UL` والتي تعني تعريف الثابت `F_CPU` بقيمة `= 1000000` وهذه السرعة تعتبر سرعة المعالج الداخلي (التردد الذي يعمل به المعالج داخل المتحكم الدقيق). يجب دائماً أن نضع هذه العبارة في بداية أي برنامج للمتحكمات الدقيقة، وكما سنرى في الفصول المتقدمة أنه يمكننا تغيير هذا الرقم وكذلك سرعة المعالج من 1 ميجاهرتز إلى 16 ميجاهرتز.

ملاحظة: الرمز `UL` في العبارة `#define F_CPU 1000000UL` يعني كلمة `unsigned long` وتستخدم للتحكم في حجم الثوابت والمتغيرات كما سنرى في الفصل القادم.

في السطر الثاني والثالث. قمنا باستدعاء مكتبيتين وهما `avr/io.h` و `avr/delay.h` و يتم ذلك باستخدام الأمر الخاص باستدعاء المكتبات `#include` ثم يكتب اسم المكتبة داخل قوسين `<...>`.

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

المكتبة الأول `io.h` هي المكتبة المسئولة عن الـ GPIO والتحكم بها وكذلك تسمية كل مخرج باسم خاص به مثل `PORTA` أو `PB0` أو `PC1` ... الخ (كما سنرى بالتفصيل في التجارب القادمة).

المكتبة الثانية `delay.h` هي المسئولة عن التلاعب بالزمن وحساب الوقت الذي يمر على تشغيل المعالج وهي المكتبة التي تمكنا من إضافة تأخير زمني أو التحكم في وقت تشغيل أي مخرج.

ملاحظة: كلتي `avr` و `util` الموجودة قبل أسماء المكتبات تعبر عن أسماء "المجلدات Folders" التي تتواجد بها هذه المكتبات، حيث قامت شركة ATmel بتوزيع المكتبات على مجلدات لتسهيل عملية تصنيفها.

الجزء الثاني من البرنامج هو الدالة `Main` والتي ستحتوي بداخلها على البرنامج الحقيق الذي يتم تشغيله على المتحكم الدقيق. غالباً ما يتم تقسيم الدالة `Main` إلى جزأين كالتالي:

- الإعدادات الخاصة بالمسجلات Registers configurations



• البرنامج الذي يتم تشغيله باستمرار while loop

```
int main(void) {  
    هنا تكتب اعدادات المُسجلات //  
    while(1)  
    {  
        هنا تكتب كافة الأكواد البرمجية //  
        التي سيتم تنفيذها بصورة مستمرة على المُتحكم الدقيق //  
    }  
    return 0;  
}
```



إعدادات مسجلات الدخول والخروج الرقمي

تمتلك متحكمات AVR عدد 3 مسجلات أساسيات للتحكم في أي بورت والتي يتم ضبطها في الـ Main function أشهر هذه المسجلات هي:

DDR x → Data Direction Register.

PORT x → Port Output Register.

PIN x → Port Input Register.

DDR_x Register المسجل

DDR_x هو مسجل 8 بت يتحكم في "اتجاه البيانات" ويعتبر المسئول عن التحكم في أطراف أي بورت لعمل إما كدخل Input أو خرج Output، حرف الـ **x** في نهاية اسم المسجل (وذلك جميع المسجلات) يعبر عن أحد الرموز A,B,C,D وهي أسماء البويرات. فمثلاً DDRC هو مسجل اتجاه البيانات للبورت C والمسجل DDRA هو الخاص بالبورت A وهكذا ..

كل بت داخل هذا المسجل تتحكم في أحد الأطراف الخاصة بالبورت حيث يعبر رقم 1 عن أن هذا الطرف يعمل كخرج output أما 0 فيعبر عن أن هذا الطرف يعمل كدخل input.

عندما نضبط أحد الأطراف لعمل كخرج فهذا يعني أنه يمكن توصيل أي عنصر الكتروني بهذا الطرف والتحكم به خلال إرسال إشارات كهربائية إلى "output signal" هذه العناصر قد تكون Led, Motor, LCD, Relay, speaker ... الخ. كما سنرى في التجارب القادمة.

اما إذا جعلنا هذا الطرف يعمل كدخل عندها يمكنه استقبال إشارة كهربائية من خلال "input signal" مثل الإشارات القادمة من المفاتيح switch أو الحساسات .sensors

وكلما نرى في الجدول التالي (الخاص بالمسجل DDRA) (الخاص بالمسجل DDRA صفحة 66 من دليل بيانات ATmega16). نجد أنه يتكون من 8 بت بدءاً من البت رقم 0 إلى البت رقم

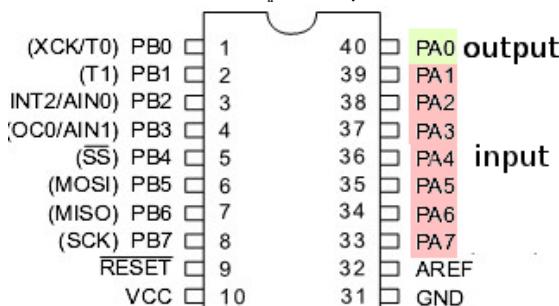
7

Port A Data Direction Register – DDRA								
Bit	7	6	5	4	3	2	1	0
Read/Write	R/W							
Initial Value	0	0	0	0	0	0	0	0

يتم التحكم في هذه البتات عبر وضع القيمة المطلوب بها مباشرة مثل أن نكتب الأمر

DDRA = 0b00000001;

هذا يعني أن نضع القيمة 00000001 داخل المسجل DDRA والتي تعني أن البت الأولى فقط = واحد أما باقي البتات = صفر مما يعني أن الطرف PA0 يعني كخرج output أما باقي الأطراف في البورت تعمل كدخل input كما في الصورة التالية:

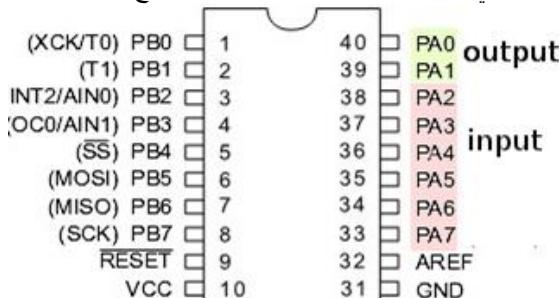


وإذا قمنا بتعديل الأمر ليصبح:

DDRA = 0b00000011;



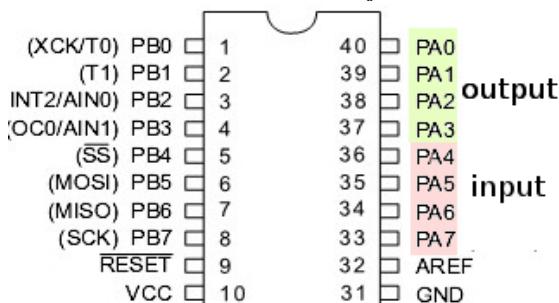
فهذا يعني أن الطرف **PA0** و **PA1** تعمل كخرج أما الأطراف من **P2** إلى **P7** تساوي صفر و تعمل كدخل.



وإذا قمنا بكتابة الأمر السابق ليصبح

DDRA = 0b00001111;

فهذا يعني أن أول 4 أطراف من البورت **PA0,1,2,3** تعمل كخرج وأخر 4 أطراف تعمل كدخل



ملاحظة: عندما نكتب رقم يبدأ ب **b0** في لغة السي مثل **b0000111000** فهذا يعني أننا نكتب رقم بالصيغة الثنائية **binary** أما عندما نكتب رقم يبدأ ب **x0** مثل **xff0** فهذا يعني أن الرقم مكتوب بالصيغة **hexadecimal**. ويمكنك أن تتعرف أكثر على أنواع الصيغ عبر قراءة الملحق المسمى "أساسيات الأنظمة الرقمية".

أيضاً لاحظ أن جميع المسجلات والأرقام الثنائية تبدأ العد من اليمين إلى اليسار وهذا يعني أن **الـ LSB** (أول بت) هي البت الموجود على الطرف الأيمن من الرقم **bXXXXXXXXX0** أما **الـ MSB** (آخر بت) فهي الموجود على الطرف الأيسر بعد الحرف **b** مباشرة.

كما هو ملاحظ في الصورة الخاصة بالمسجل **DDRA** سنجد هناك كلمة تسمى **Initial value** والتي تعني القيم الافتراضية لكل البتات والتي تساوي صفر مما يعني أن جميع الأطراف تعمل بصورة افتراضية كدخل.

أيضاً سنجد أن أسفل كل بت كلمة **Read/Write** والتي تعني أنه يمكنك تعديل محتوى هذا المسجل **write** كما فعلنا في الأمر **DDRA=0b00000001** أو يمكنك قراءة محتواه **Read** وستوضح هذه الخاصية بالتفصيل في الفصل القادم حيث سنقوم بقراءة هذه المسجلات.

المُسِّجل **PORTx Register**

يتحكم المسجل **PORTx** في الخرج الرقمي لأي طرف، فمثلاً عندما قمنا بتوسيط الدايمود الضوئي على الطرف **PA0** قمنا بتشغيله وإطفاءه باستخدام هذا المسجل، ومثل **الـ DDRx** فإنه يمتلك 8 بت كل بت منهم تتحكم في أحد الأطراف لكل بورت. الصورة التالية مثال على المسجل **PORTA** (صفحة 66 من دليل البيانات).



Port A Data Register – PORTA								
Bit	7	6	5	4	3	2	1	0
Read/Write	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
Initial Value	R/W							

كل بت في هذا المسجل تحمل إما القيمة 0 - LOW أو القيمة 1 - HIGH وعندما يتم وضع القيمة = 1 فهذا يعني أن المُتحكم سيخرج إشارة كهربية logic HIGH والتي ستكون 5 فولت (أو نفس قيمة فارق الجهد الذي يعمل به المُتحكم). أما إذا كانت القيمة = صفر فهذا يعني أن الطرف سيكون LOW أو 0 فولت وسينتقل إلى وضع الـ sink mode (سيتم شرح بالتفصيل في الفصل التالي).

في المثال السابق استخدمنا مجموعة الأوامر

```
while(1)
{
    PORTA = 0b00000001;
    _delay_ms(1000);

    PORTA = 0b00000000;
    _delay_ms(1000);
}
```

هذه الأوامر كانت تستخدم للتلاعب بالقيم الخاصة بالمسجل PORTA كالتالي:

الأمر PORTA = 0b00000001 يعني تغيير قيمة البت الخاصة بالطرف PA0 لتساوي 1 أما باقي البتات تساوي 0 وهذا يعني إخراج إشارة كهربية بقيمة HIGH على الطرف PA0 والتي ستجعل الدايويد الضوئي المتصل بهذا الطرف يضيء نتيجة الإشارة الكهربية أما باقي الأطراف تكون LOW (0 volt)

الأمر _delay_ms(1000) يعني أن المُتحكم الدقيق سيتظر 1000 ملي ثانية قبل تنفيذ الأمر التالي (لاحظ أن 1000 ملي ثانية = 1 ثانية).

الأمر PORTA = 0b00000000 مثل الأمر السابق ولكن باختلاف أن جميع البتات الآن أصبحت تساوي صفر بما في ذلك البت الخاصة بالطرف PA0 مما سيجل هذا الطرف يساوي (0 volt)

ثم يأتي الأمر _delay_ms(1000) ليجعل المُتحكم الدقيق ينتظر 1000 ملي ثانية مرة أخرى قبل أن يعاد تنفيذ جميع الأوامر السابقة بسبب الدالة (1)

نظرة عامة على المثال الأول

الكود التالي هو نفس المثال بعد إضافة تعليقات على كل سطر تشرح وظيفته.

```
#define F_CPU 1000000UL          // تحديد سرعة المعالج
#include <avr/io.h>              // استدعاء المكتبات البرمجية
#include <avr/delay.h>
```



```

int main(void)
{
    DDRA = 0b00000001;           تفعيل الطرف الأول ليعمل كخرج // 

while(1)           استمر في هذا البرنامج إلى ما لا نهاية // 
{
    PORTA = 0b00000001;           قم بتشغيل البت الأولى // 
    _delay_ms(1000);           انتظر 1000 ملي ثانية // 
    PORTA = 0b00000000;           قم بطفاء البت الأولى // 
    _delay_ms(1000);           انتظر 1000 ملي ثاني // 

}

return 0;           نهاية البرنامج // 
}

```

ملاحظة: العلامات // أو العلامات /* */ تعني أن الكلام المكتوب هو تعليق comment ولا يحتسب ضمن أكواد البرنامج، ويعتبر استخدام التعليقات أمر هام جداً لتوضيح الأكواد. لذا أنصحك أن تكتب دائماً تعليق على كل سطر برمجي أو دالة في برنامجك.

العديد من محترفي البرمجة قد يقومون بكتابة التعليقات حتى قبل البدء في كتابة الأكواد نفسها ويساعدهم ذلك على تنظيم الأفكار وتحديد ما يجب أن يكتب بصورة منتظمة، لذا احرص دائماً على توضيح وشرح كل سطر برمجي تكتبه باستخدام comment يسبق هذا السطر

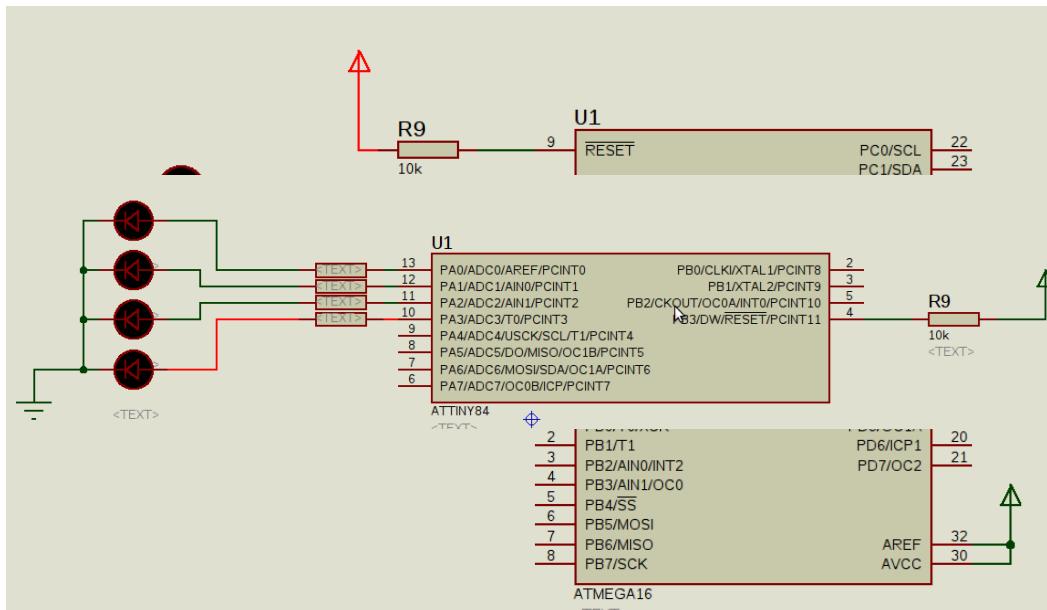


4.3 المثال الثاني: استخدام 4 دايوود ضوئي

في هذا المثال سنستخدم 4 دايوادت ضوئية حيث سقوم بتطوير الكود المستخدم في المثال الأول ليعمل بعدد **4 من الدايوادات ضوئية**. وكما هو موضح في الصورة التالية نجد الدايوادات متصلة على الأطراف من PA0 إلى PA3 سواء كنت تستخدم 16 ATmega أو ATtiny84.

استخدام
تكون
كالتالي:

في حالة ATTiny84
الوصلات





الكود البرمجي

```

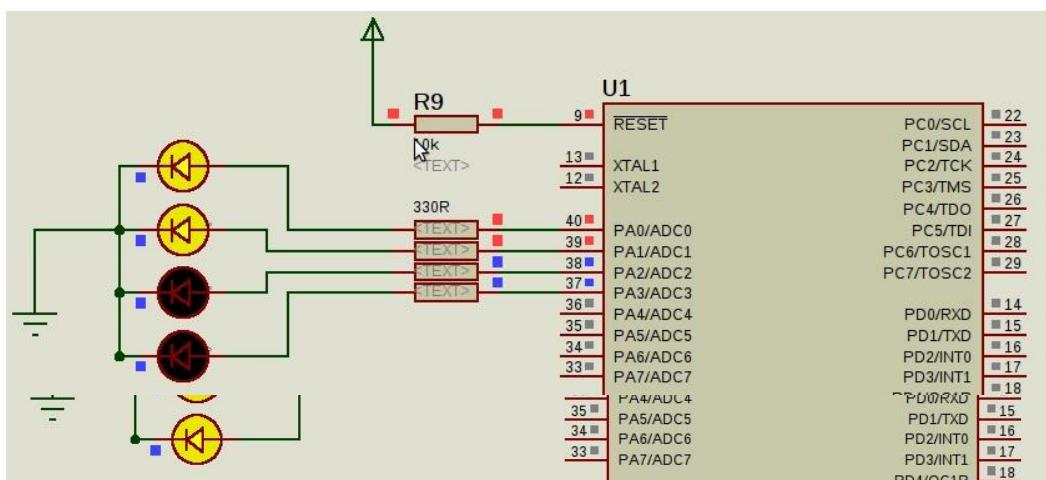
#define F_CPU 1000000UL // تحديد سرعة المعالج
#include <avr/io.h>
#include <avr/delay.h> // استدعاء المكتبات البرمجية
int main(void)
{
    DDRA = 0b00001111; // تفعيل أول 4 أطراف كمخرج

    while(1)
    {
        PORTA = 0b00000001; // شغل الطرف الأول
        _delay_ms(500); // انتظر نصف ثانية
        PORTA = 0b00000011; // شغل الطرف الثاني مع الأول
        _delay_ms(500); // انتظر نصف ثانية
        PORTA = 0b00000111; // شغل الطرف الأول، الثاني والثالث
        _delay_ms(500); // انتظر نصف ثانية
        PORTA = 0b00001111; // شغل أول أربعة أطراف من البورت
        _delay_ms(500); // انتظر نصف ثانية
    }
    return 0;
}

```

شرح الكود

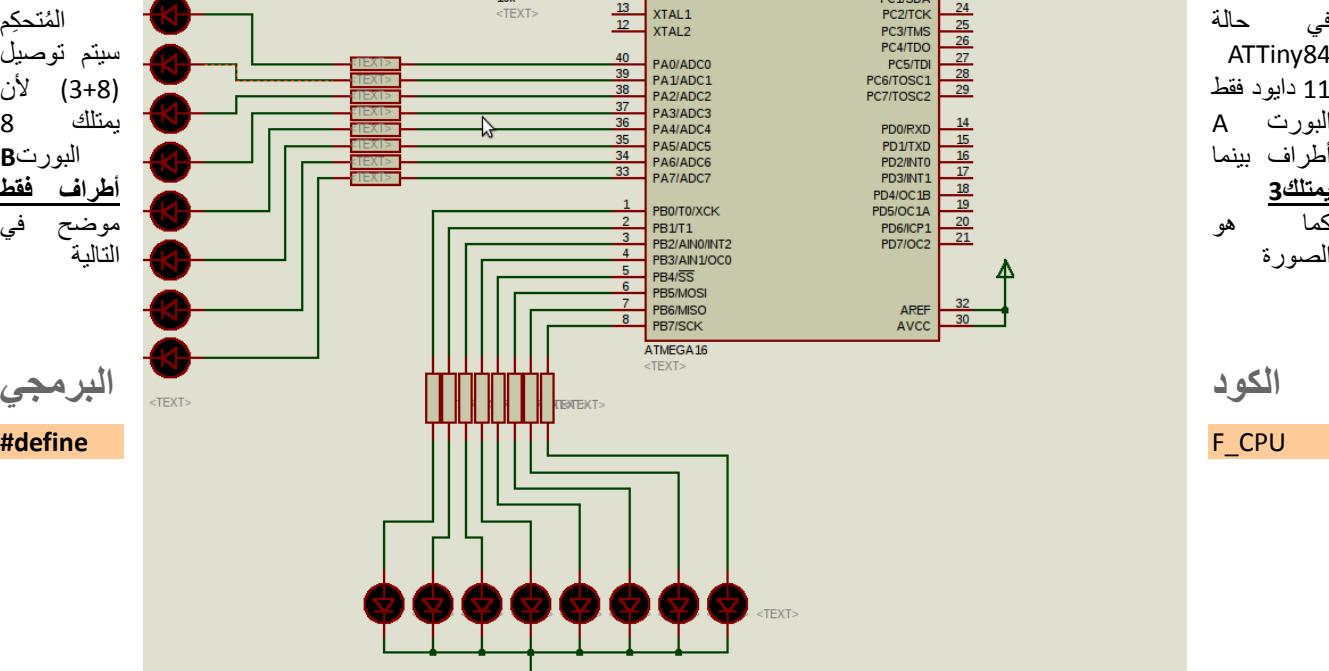
كما نرى في المثال بالأعلى، يعتبر مطابق لفنس المثال الأول باختلاف أننا استخدمنا 4 دايرودات وبالتالي قمنا بضبط المسجل DDRA ليجعل أول 4 اطراف PA0، PA1، PA2، PA3 تعمل كخرج بثليثي حيث يشغل الدايرودات الأربع. ثم يأتي الكود المكتوب داخل while (1) والذي يقوم بتنغير محتوى PORTA بصورة تصاعدية بحيث يشغل دايرود واحد كل 500 ملي ثانية (نصف ثانية). والصور التالية توضح ما سيحدث للدايرودات.





4.4 المثال الثالث: تشغيل جميع أطراف PortA, Port B

في هذا المثال سنقوم بتوصيل 16 دايدون على جميع أطراف البورت A و B بحيث يتصل 8 دايدونات ضوئية لكل بورت كما هو موضح بالصورة التالية



البرمجي

#define

```

1000000UL          // تحديد سرعة المعالج
#include <avr/io.h>  // استدعاء المكتبات البرمجية
#include <avr/delay.h>

int main(void)
{
  DDRB = 0b11111111; // تفعيل جميع أطراف Port B كخرج
  DDRA = 0b11111111; // تفعيل جميع أطراف Port A كخرج

  while(1)
  {
    PORTB = 0b11111111; // شغل جميع أطراف Port B
    PORTA = 0b00000000; // إطفاء جميع أطراف Port A
    _delay_ms(500);     // انتظر نصف ثانية

    PORTB = 0b00000000; // إطفاء جميع أطراف Port B
    PORTA = 0b11111111; // شغل جميع أطراف Port A
    _delay_ms(500);     // انتظر نصف ثانية
  }
  return 0;
}

```

في حالة ATTiny84
11 دايدون فقط
A البورت بينما
3 يمتلك
كما هو
الصورة

ال코드

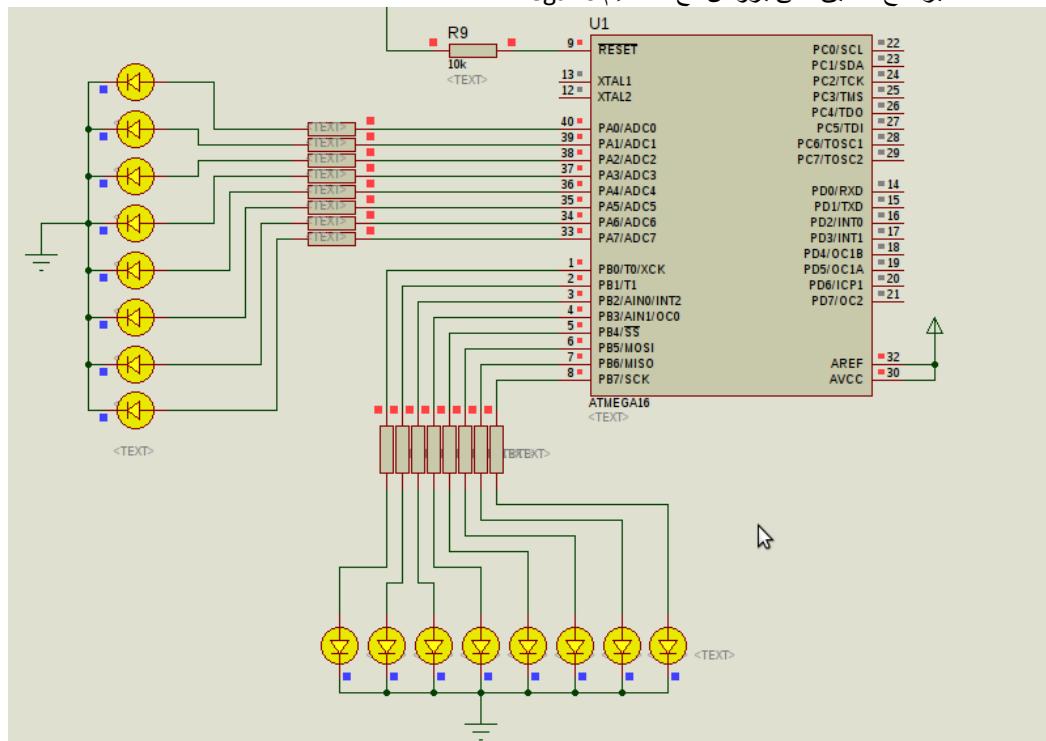
F_CPU



شرح الكود

في البرنامج السابق قمنا باستخدام المسجلين DDRA و DDRB لتشغيل جميع أطراف الپورت A والپورت B لعمل كخرج. ثم قمنا باستخدام المسجلين PORTA و PORTB لتشغيل جميع الـ Leds على هذه الأطراف لمدة نصف ثانية ثم إطفاها لنصف ثانية. وهكذا إلى ما لا نهاية.

الصورة التالية تمثل محاكاة البرنامج السابق على بروتوكول المتحكم ATmega16



إضافية

بتشغيل 8 دايو دات
ضوئية على جميع أطراف
الپورت A وجعلها
تصفيء
بالترتيب
التالي مع
تأخير ربع
ثانية فقط بين
كل أمر. (لا
تنسى أن
جميع أطراف
الپورت
ستعمل
ستعمل
خرج)
00000001
00000011
00000111
00001111
10000000
11000000
11100000
11110000

تمارين

قم

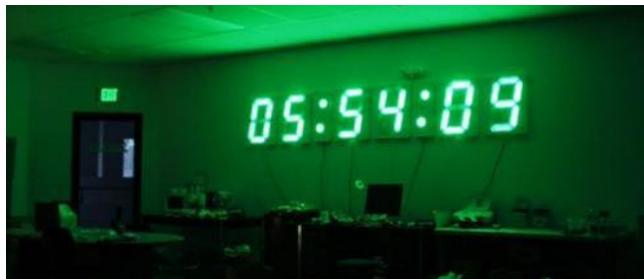
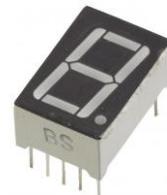
ما هو أقصى عدد من الدايو دات الضوئية يمكن توصيله بالمتحكم ATtiny84 و المتحكم ATmega16 ؟



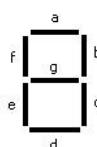
4.5 المثال الرابع: تشغيل المقاطعة السباعية7 segment

"المقاطعة السباعية - تتطق سيفين سيمجنت" segment7 وهي عبارة عن مستطيل صغير يحتوي على 7 مقاطع مضيئة باستخدام دايدادات ضوئية (متوفرة باللون الأحمر والأخضر والأزرق). وتستخدم في عرض الأرقام وبعض حروف اللغة الإنجليزية.

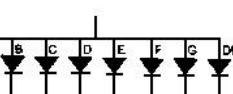
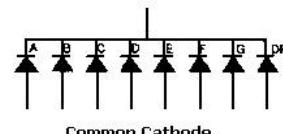
تتوفر هذه القطعة الإلكترونية في الأسواق بمختلف الأحجام فمنها ماهو صغير جداً مثل المستخدمة في الساعات الرقمية الرخيصة ومنها ماهو كبير الحجم مثل المستخدمة في إشارات المرور (لوحة المضيئة التي تعرض الوقت المتبقى لتفتح إشارة المرور).



ت تكون السيفين سيمجنت من 7 دايدادات ضوئية متصلة ببعضها البعض إما عن طريق توصيل الطرف الموجب وتسمى common anode أو عن طريق توصيل الطرف السالب وتسمى common cathode (سنتخدم في التجارب التالية النوع common cathode). ويسمى كل دايد ضوئي بأحد حروف الأبجدية الإنجليزية A,B,C,D,E,F,G كما هو موضح في الصورة التالية:



7-segment display with segment identification.



0123456789

Equivalent circuit, with decimal point.

لتشغيل هذا العنصر الإلكتروني سنقوم بتوصيل الأطراف السبعة a,b,c,d,e,f,g بأحد البورتات في المُتحكم الدقيق. (سنتخدم البورت A).

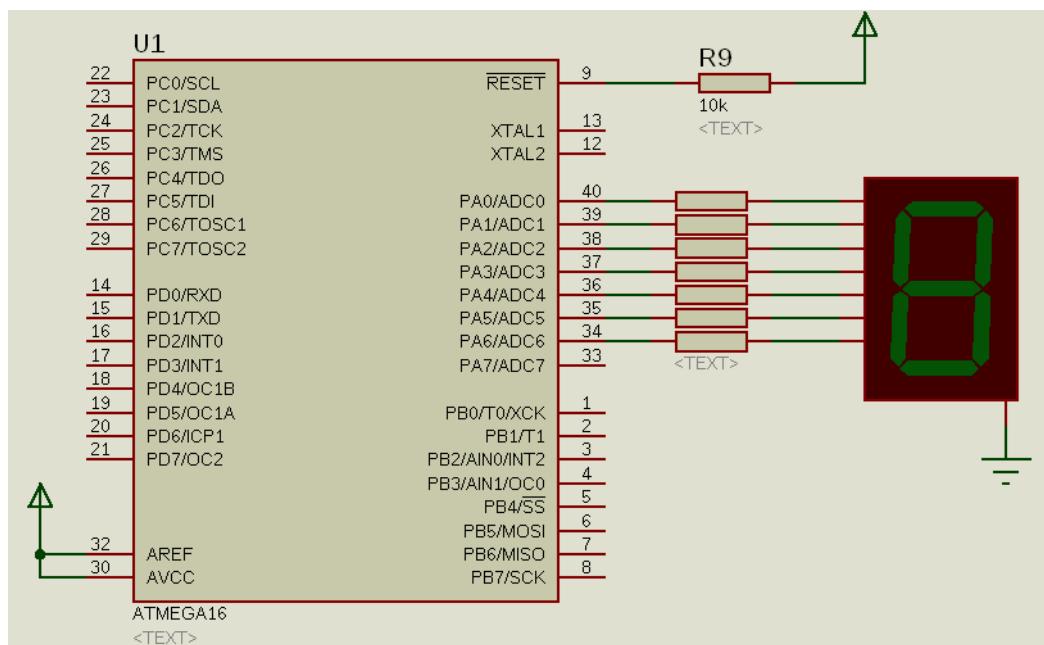
ملاحظة: بعض السيفن-سيجمنت المتوفرة في الأسواق (خاصة صغيرة الحجم) تحتوي على طرف إضافي وهو دايد ضوئي صغير موجود على الجانب الأيمن السفلي ويستخدم في عرض الفاصلة العشرية (.) لكن برنامج بروتوكول يحتوي على هذا الدايد الضوئي لذا لن نستخدمه في المحاكاة.

لعرض أي رقم من الأرقام العشرية سنستخدم الجدول التالي والذي يوضح الحالة التي يجب أن يكون عليها كل دايد ضوئي حتى يتم عرض رقم معين.



DIGIT	LEDs TO GLOW						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

الشكل التالي يوضح طريقة توصيل المُتحكِّم ATmega16 بالسيفين-سيجمنت على البرت A. وسنقوم بكتابة كود بسيط يعرض الأرقام من 0 إلى 9 بالترتيب وبتأخير زمني 1 ثانية بين كل رقم.



الكود البرمجي

```

#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRA = 0b11111111;

    while(1)
    {
        PORTA = 0b00111111; // Number 0
        _delay_ms(1000);
        PORTA = 0b00110000; // Number 1
        _delay_ms(1000);
        PORTA = 0b01011011; // Number 2
        _delay_ms(1000);
        PORTA = 0b01001111; // Number 3
        _delay_ms(1000);
        PORTA = 0b01100110; // Number 4
        _delay_ms(1000);
        PORTA = 0b01101101; // Number 5
        _delay_ms(1000);
    }
}

```



```

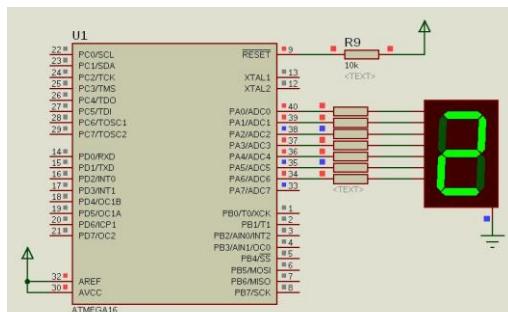
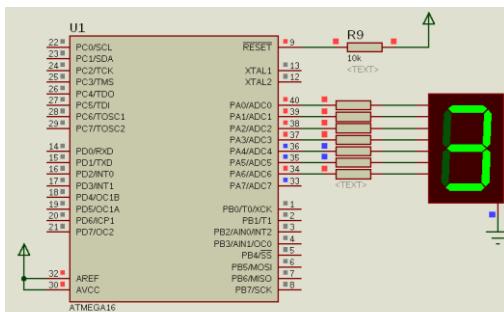
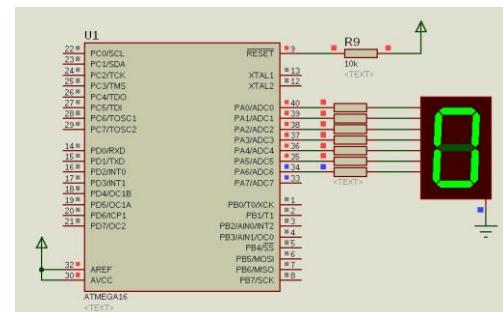
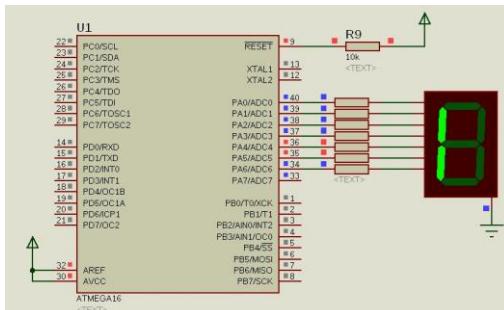
PORTA = 0b01111101; // Number 6
    _delay_ms(1000);
PORTA = 0b00000011; // Number 7
    _delay_ms(1000);
PORTA = 0b11111111; // Number 8
    _delay_ms(1000);
PORTA = 0b01101111; // Number 9
    _delay_ms(1000);

}

return 0;
}

```

الصور التالية تمثل المحاكاة بعد ترجمة الكود السابق.



شرح الكود

البرنامـج السابـق قـام بـتشغـيل السـيفـن سـيـجـمـنـت بـحيـث تـعرـض جـمـيع الأـرـقـام مـن 0 إـلـى 9 بـصـورـة مـتـابـعـة وـذـلـك عـبـر كـاتـبـة قـيـمة الرـقـم المـطـلـوب دـاخـل المـسـجـل.

هـنـاك مـلـاحـظـة هـامـة حـول هـذـا الكـوـد وـهـي أـنـ الـأـرـقـام الـتـي يـتـم وـضـعـها دـاخـل المـسـجـل **PORTA** تـعـتـبر مـعـكـوسـة عـنـ الـجـوـلـ. المـكـتـوب بـالـأـعـلـى وـذـلـك بـسـبـب أـنـ الـأـطـرـافـ الـتـي قـمـنـا بـتـوـصـيـلـهـا عـلـى بـرـنـامـج بـرـوـتـسـ تم عـكـسـهـا فـبـدـلـاً مـن تـوـصـيـلـ



تم توصيلها a,b,c,d,e,f,g

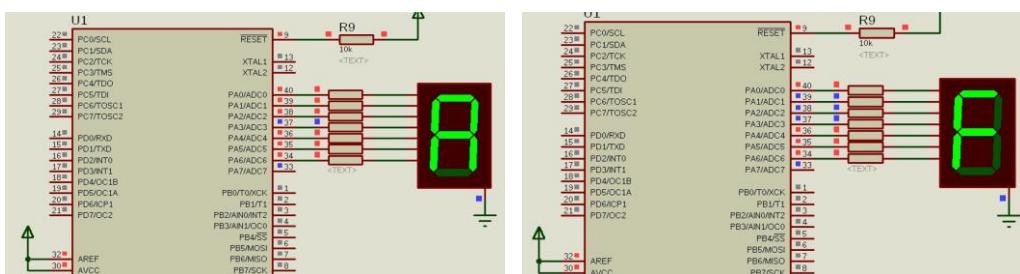
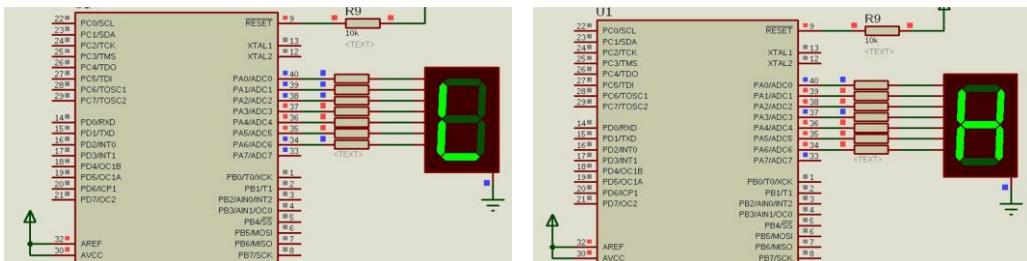
أيضاً يمكنك كتابة بعض الحروف الإنجليزية البسيطة مثل A,C,F,E,H,L كل ما عليك فعله هو إضافة الجزء التالي لل코드 بالأعلى (داخل الـ (while loop).

```

PORTA = 0b01110111; // Letter A
_delay_ms(1000);
PORTA = 0b00111001; // Letter C
_delay_ms(1000);
PORTA = 0b01110001; // Letter F
_delay_ms(1000);
PORTA = 0b00111000; // Letter L
_delay_ms(1000);
PORTA = 0b01110110; // Letter H
_delay_ms(1000);

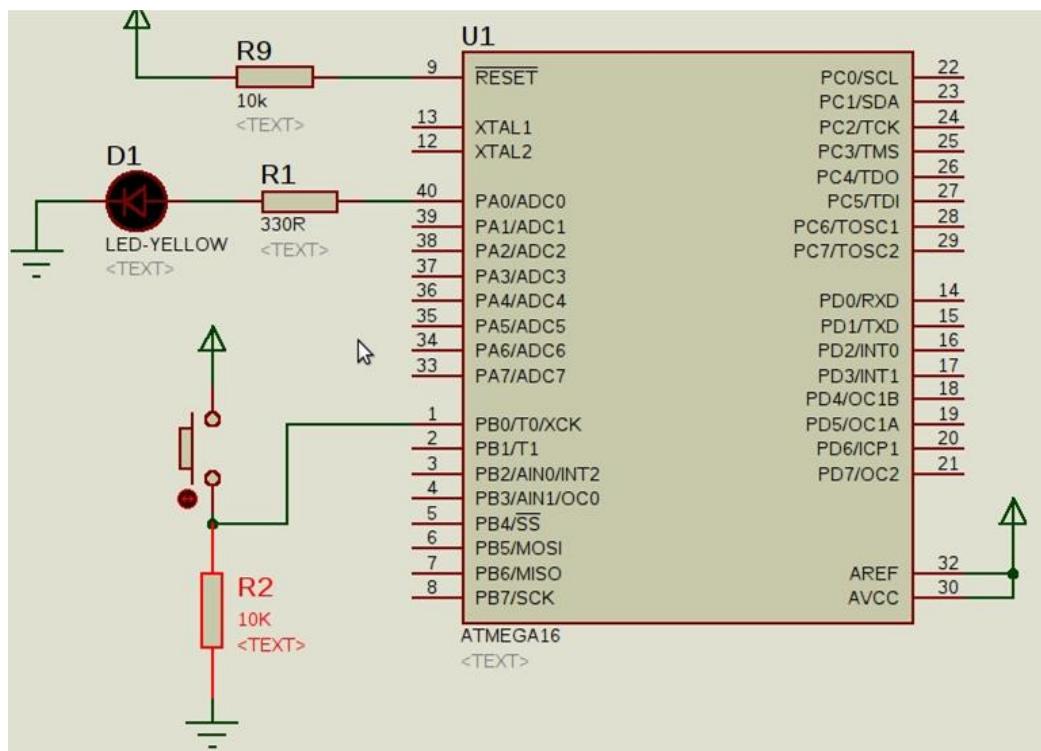
```

الصور التالية توضح عرض حروف الأبجدية الإنجليزية باستخدام السيفن-سيفون

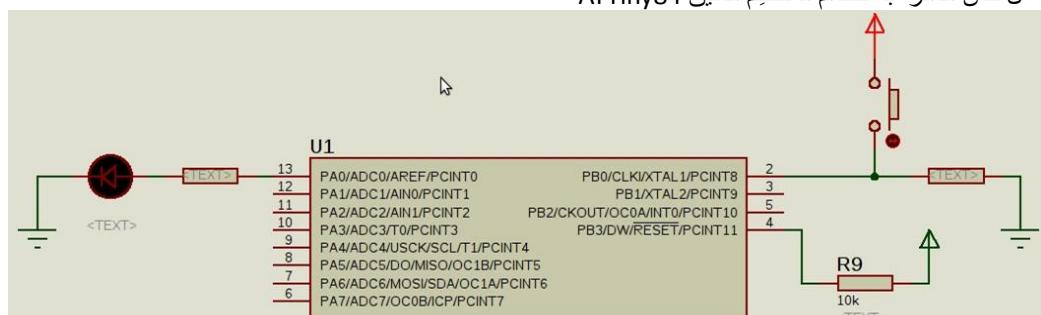


4.6 المثال الخامس: قراءة الدخل الرقمي

في هذا المثال سنتعرف على طرق قراءة الدخل الرقمي Digital Inputs وذلك عبر مجموعة من التجارب باستخدام المفاتيح Switchs. سيتم استخدام مفتاح الضغط Push button مع مقاومة 10 كيلو اوم متصلة على الأرضي كدخل للطرف PB0 وذلك للتحكم بتشغيل دايدود ضوئي متصل على الطرف PA0 كما هو موضح بالصورة التالية:



لعمل نفس الدائرة باستخدام المُتحكم الدقيق ATTiny84



الكود البرمجي

```

#define F_CPU 1000000UL
#include <avr/io.h>

int main(void)
{
    DDRA = 0b00000001;
    while(1)
    {
        if (PINB == 0b00000001)
    }
}

```



```

{PORTA = 0b00000001;}
else
{PORTA = 0b00000000;}
}
return 0;
}

```

قم بترجمة الكود ثم استخدم ملف الهيكس لمحاكاة المشروع، حيث ستجد أنه عند الضغط على المفتاح سنجد الدايوه الضوئي يبدأ بالإضاءة **Buttn press** وعند ترك المفتاح **BUTTON release** ينطفئ الدايوه الضوئي.

شرح الكود

بصورة افتراضية تعمل جميع أطراف المُتحكمات الدقيقة كدخل **Input port** (ليس AVR فحسب وإنما معظم المُتحكمات الدقيقة من مختلف الشركات) لذا نجد الكود السابق لا يقوم بضبط أطراف البورت B لعمل كدخل، ومع ذلك يمكنك كتابة الأمر **DDRB = 0b00000000** للتأكد أن أطراف المُتحكم تعمل كدخل.

يقوم المُتحكم الدقيق بصورة تلقائية بقراءة محتويات جميع الأطراف ويخزنها في المُسجل **x** PIN (حيث تمثل **x** اسم البورت مثل **A,B,C,D ... إلخ**) فمثلاً المُسجل PINB يقوم بتسجيل قيمة الجهد الداخل على جميع أطراف البورت B بصورة مستمرة ويتم تحديث هذا المُسجل بنفس سرعة عمل المُتحكم الدقيق فمثلاً إذا كان المُتحكم يعمل بسرعة 1 ميجاهرتز فهذا يعني أن المُسجل PINB يتم تحديثه مليون مرة في الثانية وفي كل مرة يقوم بقراءة جميع أطراف البورت B.

يحتوي المُسجل PINx على 8 باتات كل بت تمثل قراءة الجهد على أحد أطراف البورت، فمثلاً المُسجل PINB نجد أنه يتكون من البتات التالية (جميع جداول PINx متوفرة في دليل البيانات).

Port B Input Pins Address – PINB		Bit	7	6	5	4	3	2	1	0	PINB
Read/Write			PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	
Initial Value			R	R	R	R	R	R	R	R	
			N/A								

البت رقم صفر PINB0 تمثل قراءة الطرف PB0 والبت رقم 1 تمثل قراءة الطرف PB1 والبت رقم 2 تمثل قراءة الطرف PB2 ... إلخ.

ملاحظة: المُسجل PINB وجميع المُسجلات PINx من نوع **Read only** مما يعني أنه يمكنك أن تقرأ منها فقط ولا يمكنك أن تغير محتواها بنفسك بكتابة أي كود برمجية حيث يتم تحديث المُسجلات تلقائياً.

في حالة عدم تطبيق أي جهد على أطراف المُتحكم تكون قيمة البتات = صفر ويتم تغيير هذه القيمة عند تطبيق جهد على الطرف المואزي لكل بت. مثلاً لو قمنا بتطبيق جهد 5 فولت على البت PB0 و PB1 و PB2 سنجد أن قيمة المُسجل PINB تساوي 0b000001110 وإذا قمنا بتطبيق جهد على جميع البتات سنجد القيمة أصبحت 0b111111110 وهذا ...

في البرنامج السابق استخدمنا الجملة الشرطة (condition) **if** لتشغيل الدايوه الضوئي عند الضغط على الزر حيث كتبنا الأمر.

```

if(PINB == 0b00000001)
{PORTA = 0b00000001;}

```

والذي يعني أنه إذا كانت قيمة المُسجل PINB تساوي 0b00000010 (يعني تم الضغط على الزر المتصل بـ PB0) قم بتشغيل الدايوه الضوئي عبر الأمر **PORTA = 0b00000001**

```
else
```



{PORTA = 0b00000000;}

وفي حالة عدم تطبيق هذا الشرط else قم بإطفاء daiyod عبر PORTA = 0b00000000

Pull Up & Pull Down Resistor 4.7

من أشهر الكلمات التي قد تسمعها في عالم الإلكترونيات الرقمية هي المقاومة رافعة الجهد أو خافضة الجهد Pull-Up & Pull-Down هذه المقاومات التي تتراوح قيمتها بين 2.2 كيلوأوم إلى 10 كيلوأوم تستخدم بصورة أساسية في دخل الأنظمة الرقمية Digital Inputs بداية من البوابات المنطقية البسيطة مثل AND, OR, NOT وانتهاءً بالمتاحكمات الدقيقة.

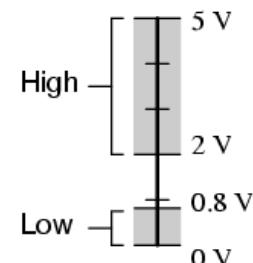
إذًا لماذا استخدامنا هذا النوع من المقاومات مثل المقاومة الـ 10 كيلو أوم مع المفتاح في المثال السابق؟ ولماذا لم نوصل المفتاح بالـ VCC مباشرةً؟ هناك 3 أسباب لهذا الأمر..

الاستخدام الأول (لغاء الدخل العام): جميع المكونات الإلكترونية الرقمية التي تتعامل بالصفر والواحد تعاني من مشكلة خطيرة جداً تسمى المنطقة العائمة Nosie Margin أو Floating Area (نطاق الشوشرة) هذه المنطقة هي فارق الجهد بين الـ Logic (0) والـ Logic (1) وتستخدم المقاومات pull-up or down في حل هذه المشكلة.

والسبب في ذلك أن معظم المكونات الإلكترونية تعتبر أي جهد دخل input voltage بين صفر و 0.8 فولت هو 0 أو كما يسمى Low بينما أي جهد بين 2 و حتى 5 فولت يعتبر 1 رقمي HIGH لكن إذا كان هناك دخل بقيمة 0.9 فولت أو 1.25 أو 1.9 أو 1.25 تحدث مشكلة المنطقة العائمة حيث لا تستطيع الإلكترونية الرقمية أن تتعرف على هذا الجهد وبالتالي لا يمكنها أن تحدد بدقة هل هو HIGH أم LOW (0) أم HIGH (1) وتنبدأ بالتنصرف العشوائي (وقد تصيب الدائرة بنوع من الجنون).

بطبيعة الحال جميع الأجهزة الإلكترونية تصدر نوع من الـ Electric noise شوشرة كهربية بما في ذلك جسد الإنسان لذا عند ترك أي طرف input يتعرض لشوشرة كهربية بفرق جهد صغير نسبياً مما يتسبب في دخول الجهاز

Acceptable TTL gate input signal levels



بوضع الـ Floating Area

وجود مقاومة متصلة بطرف الدخل والطرف الأرضي يضمن تماماً أن الطرف = صفر حتى مع وجود الشوشرة ولا يتم تغيير هذا الجهد إلا عند إدخال جهد كبير نسبياً من المفتاح مثل 5 فولت. وتكون قيمة المقاومة بين 2.2 إلى 10 كيلوأوم. يمكنك أن تجرب بنفسك تأثير المنطقة العامة عبر المثال السابق حيث يتم تركيب الدائرة كما هي على لوحة التجارب breadboard بدون مقاومة 10 كيلو. ثم حاول أن تضغط الزر وشاهد ماذا سيحدث (ستجد أن المتاحكم الدقيق يبدأ باتخاذ قرار عشوائية بمجرد أن تقترب يدك من المتاحكم).

الاستخدام الثاني (عكس الجهد الداخلي): في بعض الحالات يكون مطلوب عكس الجهد الداخل من المفتاح، في المثال السابق تم توصيل المقاومة مع المفتاح بأسلوب Pull-Down مما يجعل المفتاح يدخل جهد 5 فولت للطرف PBO عند الضغط عليه وعند ترك المفتاح يكون الجهد صفر فولت (أرضي).

Press Button = HIGH

Releas Button = LOW

يمكن استخدام المقاومة الرافعة Pull-up Resistor لعكس هذا الجهد وبذلك يصبح الجهد الأساسي للطرف PBO هو HIGH وعند ضغط المفتاح يتتحول هذا الجهد إلى صفر (أرضي).

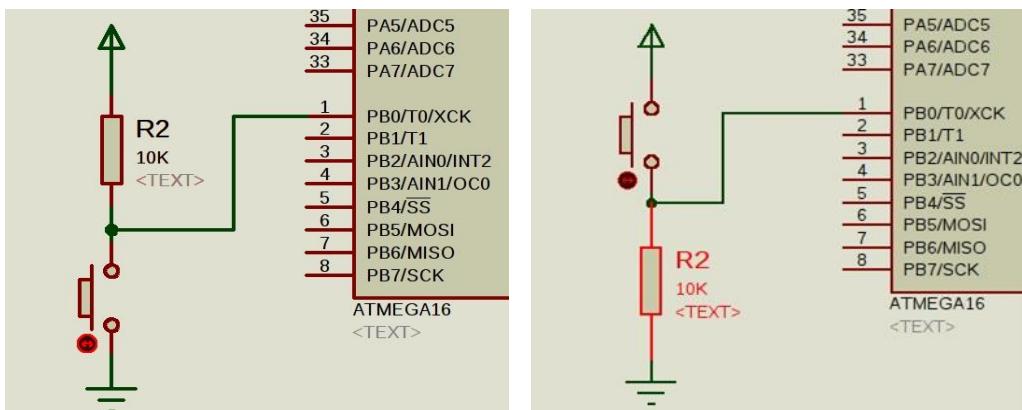
Press Button = LOW

Releas Button = HIGH

يتم توصيل المقاومة الرافعة Pull-up بحيث ينعكس مكانها بين الـ VCC والمفتاح والصور التالية توضح الفرق بين Pull-Down و Pull-UP وال

Pull-Up

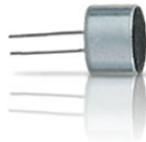
Pull-Down



ملاحظة: قد تستخدم مقاومات الـ Pull-Up & Pull-Down مع بروتوكولات الاتصالات مثل I2C لعكس جهد النبضات
ويتم استخدام مقاومات بقيمة تتراوح بين 2.2 إلى 4.7 كيلوأوم



الاستخدام الثالث (محول تيار إلى جهد): بعض الحساسات الشهيرة مثل الميكروفون Microphone أو المقاومة الضوئية LDR أو الحرارية NTC تقوم بتحويل الحرارة أو الضوء إلى تغير في التيار الكهربائي وليس تغير في الجهد مما يمثل مشكلة في فهم هذه الحساسات. حيث نجد أن جميع المتحكمات الدقيقة التي تحتوي على ADC يمكنها قراءة تغير الجهد فقط ولا تستطيع التعرف على التيار الكهربائي المتغير لذا يتم استخدام مقاومة Pull-Up أو Pull-Down لتحويل التيار المتغير إلى جهد.

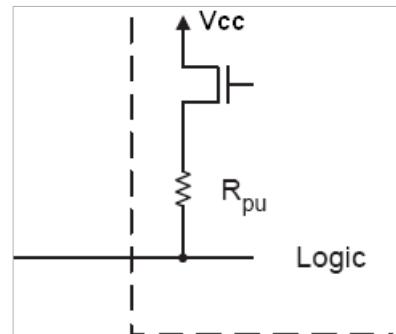




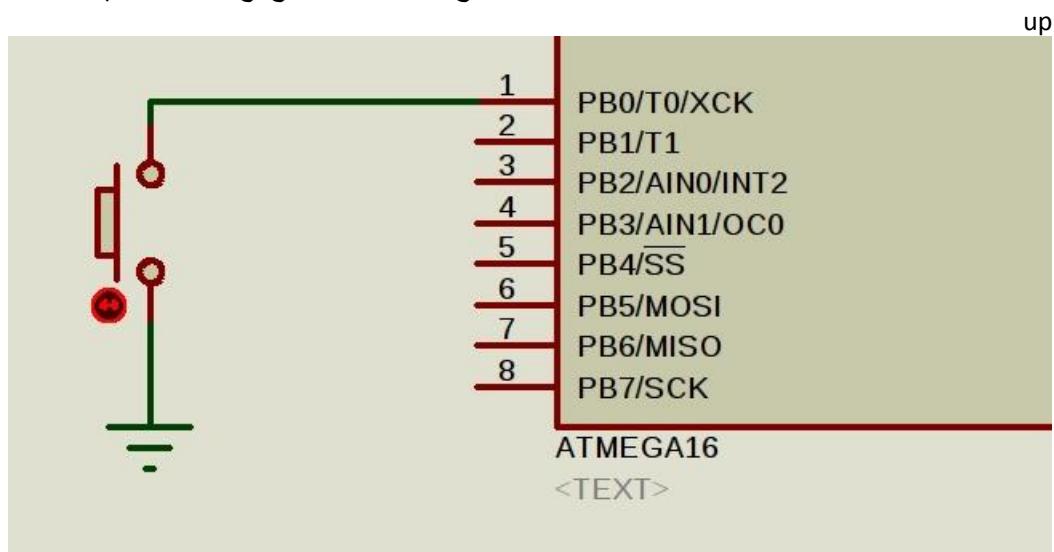
Internal Pull-Up 4.8 خاصية الـ

تمتلك معظم مُتحكمات الـ AVR خاصية جميلة جداً وهي أن أطراف البورتات تمتلك (مقاومة الرفع الداخلية internal pull-up) هذه المقاومة تجعلك تستخدم المفاتيح التي تريدها بدون أي مقاومات إضافية. الشكل التالي يوضح تركيب مقاومة الرفع الداخلية وهي عبارة عن ترانزستور + مقاومة R_{pu} حيث يتحكم الترانزستور في تفعيل المقاومة أو إلغائها.

بصورة افتراضية يتم إلغاء تفعيل هذه المقاومة ويجب عليك أن تشغلاها بنفسك وذلك عبر جعل الطرف المطلوب يعمل كدخل input ثم الكتابة في المُسجل PORTX لتفعيل هذه المقاومة لاحظ أن المُسجل PORTX لن يعمل لكتابه قيم الخرج ولكن هنا سيسخدم تفعيل الترانزستور الخاص بمقاومة الـ pull-up فقط. أيضاً تذكر أنه بعد تفعيل المقاومة يصبح تأثير الضغط على المفتاح معكوس مما يعني أن قراءة PINx تصبح 0 في حالة الضغط على المفتاح وتصبح 1 ترك المفتاح.



الصورة التالية توضح شكل توصيل المفتاح مع تفعيل الـ internal pull-up

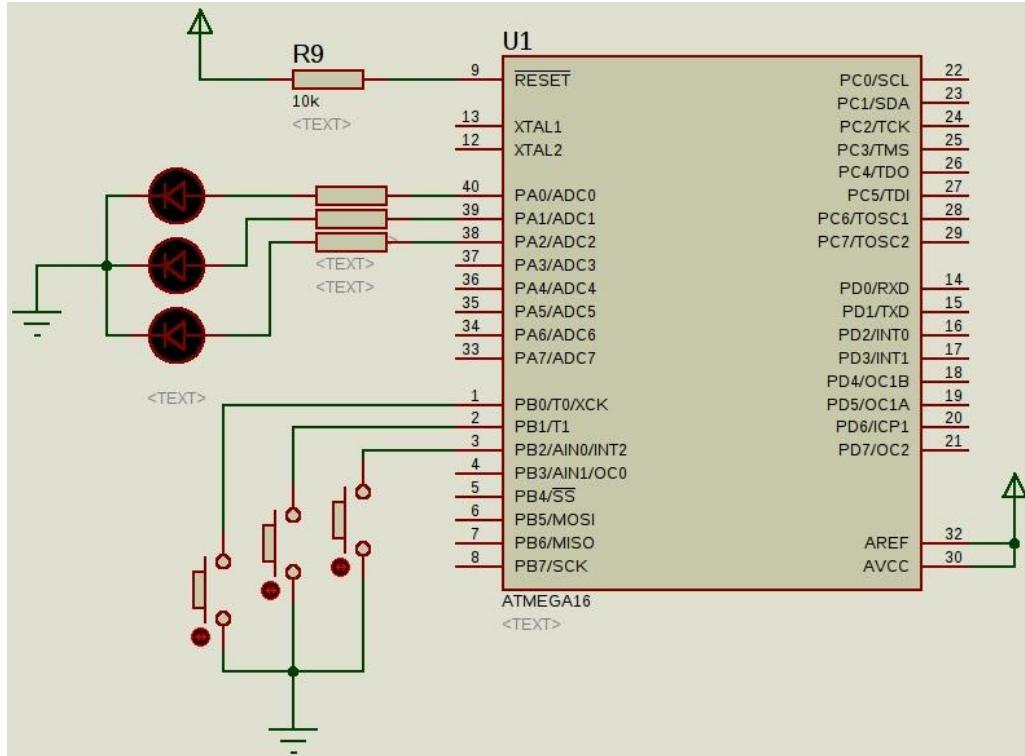




4.9 المثال السادس: تشغيل 3 دايوسات + 3 مفاتيح

في هذا المثال سنستخدم 3 مفاتيح مع تفعيل خاصية الـ Internal pull-up وبذلك سنوفر استخدام 3 مقاومات 10 كيلو. وسيتحكم كل مفتاح في تشغيل عدد من الدايوسات بحيث:

- إذا تم الضغط على المفتاح المتصل بالطرف PB0 يتم تشغيل دايو واحد فقط
- إذا تم الضغط على المفتاح المتصل بالطرف PB1 يتم تشغيل 2 دايو
- إذا تم الضغط على المفتاح المتصل بالطرف PB2 يتم تشغيل 3 دايو



تذكر أنه بعد تشغيل مقاومة الرفع تصبح قراءة المفتاح والمُسجل PINx معكوسة لذا سنجد القيمة الإفتراضية للمسجل PINx = 0b11111111 وعند الضغط على أي مفتاح سنتحول البت المقابل له إلى صفر، فمثلاً الضغط على المفتاح المتصل ب PB0 سيجعل قيمة PINB تساوي 0b11111100



الكود البرمجي

```

#define F_CPU 1000000UL
#include <avr/io.h>

int main(void)
{
    DDRA = 0b00000111; // تفعيل خرج لتشغيل الثلاث دايدات ضوئية

    DDRB = 0b00000000; // تأكيد أن جميع أطراف البورت تعمل كدخل
    PORTB = 0b11111111; // // تفعيل مقاومة الرفع لكل أطراف البورت B

    while(1)
    {
        if (PINB == 0b11111110) {PORTA = 0b00000001;} // Button 1 pressed
        else if (PINB == 0b11111101) {PORTA = 0b00000011;} // Button 2 pressed
        else if (PINB == 0b11111011) {PORTA = 0b00000111;} // Button 3 pressed
        else {PORTA = 0b00000000;}
    }
    return 0;
}

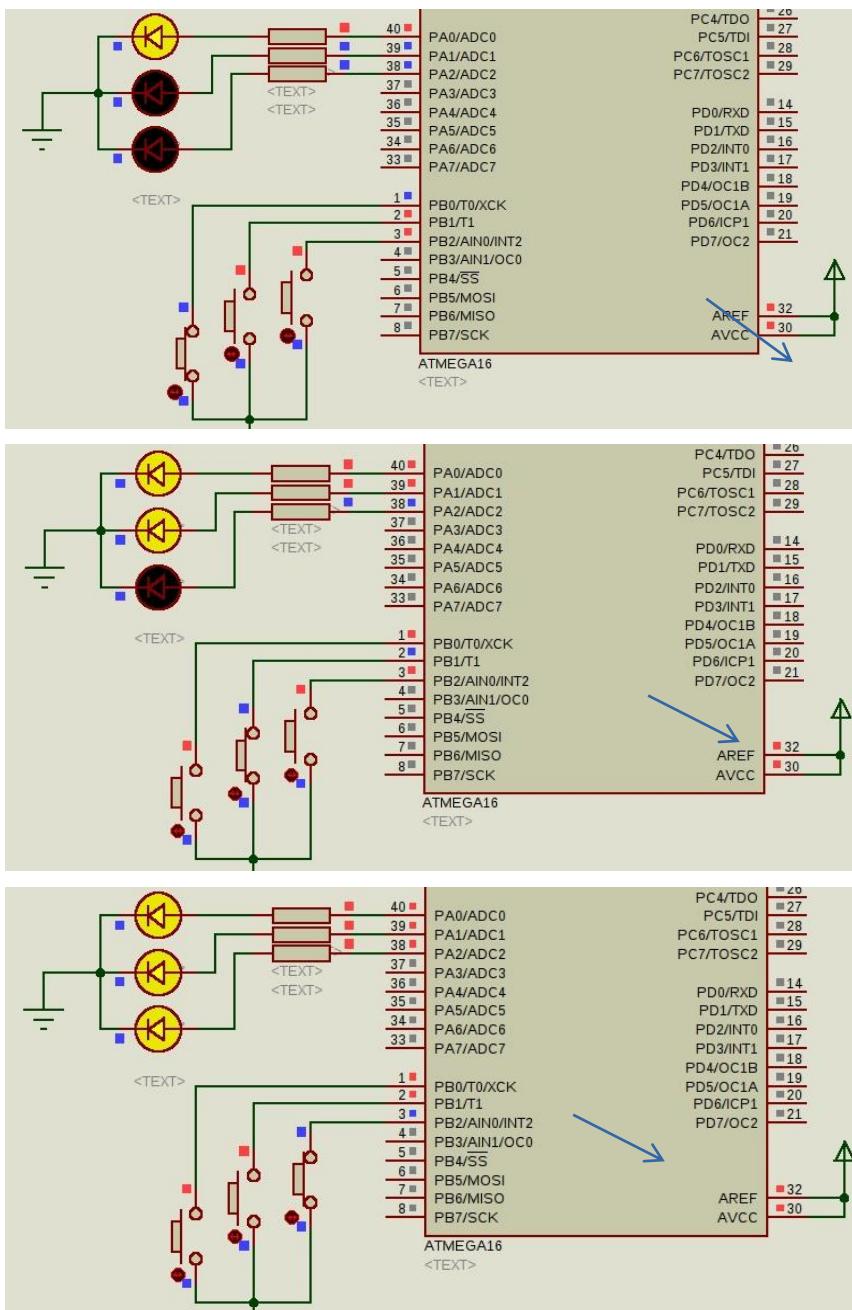
```

شرح الكود

في البداية قمنا بتشغيل جميع أطراف البورت B لعمل كدخل (تذكرة أن هذه الخطوة اختيارية وهدفها التأكيد فقط) ثم قمنا بتشغيل جميع مقاومات الـ Pull-UP على أطراف البورت B وذلك عبر الأمر `PORTB = 0b11111111`

ثم بدء البرنامج الأساسي في قراءة المُسجل PINB ومقارنته مع مجموعة من الشروط بحيث إذا كانت القيمة = `b111111100` (تم الضغط على الزر المتصل بـ PB0) يتم تشغيل الدايد المتصل بالطرف PA0

وإذا كانت القيمة تساوي `b111111010` فهذا يعني أنه تم الضغط على الزر المتصل بالطرف PB1 وسيؤدي ذلك لتشغيل الدايدات على الأطراف PA0 + PA1 ونفس الأمر مع الزر الثالث (المتصل بالطرف PB2) والذي سيشغل الدايدات الثلاثة PA0, PA1, PA2.





Bouncing effect & De-bouncing 4.10

يعرف تأثير القفز Bouncing effect بأنها ظاهرة تحدث للمفاتيح switches الميكانيكية خاصة مفاتيح الضغط Push Buttons حيث تتكون هذه المفاتيح عادة من شريحتين من الصفائح المعدنية والتي عند الضغط عليها يحدث بينهم تلامس contact. لكن بسبب طريقة صناعة هذه الصفائح فإنه عند الضغط عليها تقفز الألواح المعدنية أكثر من مرة، هذا الأمر يجعل الصفائح تقوم "بعشرات التلامسات" في الثانية الواحدة قبل أن تستقر.

تعتبر هذه الظاهرة من الأمور المزعجة لمصممي النظم المدمجة وذلك لأنها تجعل المفاتيح الميكانيكية تنتج إشارات غير مطلوبة قد تؤثر على أداء النظام المدمج ككل. من حسن الحظ أنه هناك العديد من الحلول المتوفرة لهذه الظاهرة وتسمى هذه الحلول debouncing.

الحل الأول: التأخير الزمني

هذا الحل يعتبر أبسط طريقة وتسمى software debouncing حيث تعتمد هذه الطريقة على التأكيد "مرتين" من الضغط على المفتاح مع وضع تأخير زمني بين كل مرة. الكود التالي يمثل تشغيل دايلود ضوئي عند الضغط على مفتاح (والثاني)

ال kod التقليدي لقراءة المفتاح بدون debouncing

```
if (PINB == 0b11111110) { PORTA = 0b00000001; }
    software debouncing (delay)
```

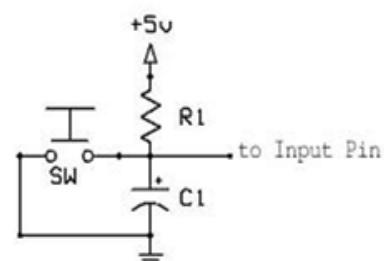
```
if (PINB == 0b11111110)
{
    delay_ms(10); // تأخير زمني
    if (PINB == 0b11111110) // إعادة التأكيد أن المفتاح مازال مضغوط
        { PORTA = 0b00000001; }
}
```

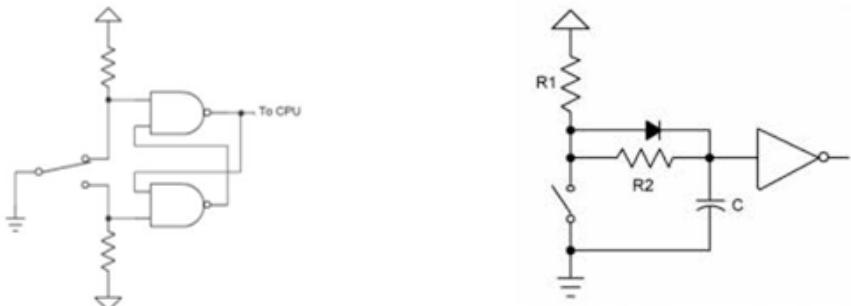
هذه الطريقة لها عيوب كثيرة جداً منها أنها لا تحل مشكلة الذبذبات الكهربائية نفسها ولكنها تتأكد فقط من ضغط المفتاح لفترة كافية حتى تنتهي تلك الذبذبات كما أنك تضطر إلى إضافة تأخير زمني للبرنامج مما قد يتسبب في تأخير استجابة النظام كل خاصية إذا كنت تقرأ أكثر من مفتاح. وفي حالة استخدام نظام تشغيل RTOS لا يمكن استخدام هذا النوع من التأخير الزمني بصورة مباشرة (سيتم شرح التأخير الزمني لنظام ROTS بالتفصيل في الفصل السابع).

الحل الثاني: استخدام فلتر (مرشح)

هذا الحل يعتمد على استخدام دائرة filter لترشيح الذبذبات الكهربائية الناتجة عن الـ bouncing effect حيث تستخدم هذه الدائرة في امتصاص الذبذبات الكهربائية وتنعى دخولها للتحكم الدقيق.

أبسط دائرة مرشح هي توصيل مكثف سيراميكي على التوازي مع المفتاح هناك دوائر أخرى أكثر تعقيداً وقوة في الترشيح مثل استخدام مقاومة 10 كيلو + دايلود + مكثف (بنفس القيمة). أو باستخدام دوائر رقمية مثل AND gates لعمل digital trigger.





على أي حال، استخدام مكثف واحد فقط كافي لفلترة الذبذبات الكهربائية الغير مرغوب فيها ولا داعي لبناء دوائر أكثر تعقيداً كما أنه أوفر من ناحية للتكلفة.

- **المميزات:** الأفضل من ناحية الأداء. متوافقة مع جميع المتحكمات الدقيقة ولا تتطلب أي تأخير زمني في الكود كما أنها لن تتسبب في مشاكل للـ RTOS أو الـ Interrupts
- **العيوب:** زيادة التكلفة بسبب استخدام مكونات إضافية مع المفتاح.



4.11 حساب المقاومة المستخدمة قبل الأحمال

الجمل Load هو أي جهاز يتصل بخرج المُتحكم الدقيق مثل الدايمود الضوئي، في الأمثلة السابقة لاحظنا وجود مقاومة قبل كل دايمود بقيمة (300 أوم)، لماذا استخدمنا هذه المقاومة وبتلك القيمة تحديداً؟

هناك سببان لهذا الأمر. الأول هو أن الدايدو الصوّي (خاصة اللون الأحمر) يفضل أن يتم تشغيله بتيار كهربائي ما بين 15 إلى 20 ملي أمبير عند تطبيق فرق جهد 5 فولت. وبالتعريض في قانون أوم الكهربائي (فرق الجهد 5 فولت - التيار 15 ملي أمبير)

$$Voltage(V) = Current(I) \times Resistance(R)$$

نجد أن المقاومة المناسبة هي 330 أوم .

إذا لم تستخدم هذه المقاومة قد تحدث العديد من المشاكل. فمثلاً إذا تعرض الدايموند الضوئي الأحمر لأكثر من 25 ملي أمبير لفترة طويلة نسبياً (ساعة أو أكثر) فإنه سيحترق وبالتالي أنت لا تزيد دائرة إلكترونية تحتاج لتغيير دايموند ضوئي كل ساعة !

ثانياً: قد يؤدي تشغيل الأحمال بدون مقاومة لفترات طويلة إلى تدمير أطراف (بورات) المتحكم الدقيق. وذلك بسبب محدودية التيار الكهربائي الذي يمكن سحبه من كل طرف، إذا نظرت إلى الصفحة الخاصة بالخصائص الكهربائية للمتحكم Electrical Characteristics (صفحة 291 دليل بيانات المتحكم ATmega16). ستجد جدولـ **Absolute values** "Absolute values" (صفحة 291 دليل بيانات المتحكم ATmega16). وهي أقصى قيم كهربائية يتحملها المتحكم الدقيق.

Electrical Characteristics

Absolute Maximum Ratings*

Operating Temperature	-55 °C to +125 °C
Storage Temperature	-65 °C to +150 °C
Voltage on any Pin except <u>RESET</u> with respect to Ground	-0.5V to V_{CC} +0.5V
Voltage on <u>RESET</u> with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	200.0 mA PDIP and 400.0 mA TQFP/MLF

***NOTICE:** Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

لاحظ السطر قبل الأخير حيث نجد DC current per I/O = 40 mA

هذا يعني أن أقصى تيار يمكن سحبه من أي طرف من أطراف المُتحكّم = 40 مللي أمبير وإذا كان الحمل يسحب أكثر من ذلك فهذا سيؤدي إلى تلف المُتحكّم الدقيق. عليك أن تذكر هذا الرقم جيداً عند تصميم أي دائرة الكترونية.

الأفضل أن تسحب الأحمال تيار = 85% من القيمة القصوى وذلك حتى تضمن أن يكون العمر الإفتراضي للمتحكم الدقيق أطول ما يمكن. فمثلاً إذا طبقنا هذه القاعدة سنجد أن التيار الآمن هو $0.85 * 40 = 34$ مللي أمبير.

هذا يعني أن أقل مقاومة يمكن توصيلها على أي طرف من أطراف المُتحكم (من قانون أوم) تساوي 147 أوم، مع ملاحظة أنه يمكن توصيل أي قيمة أعلى لأن المقاومات الأكبر ستترعرع تيار كهربائي أقل وهذا آمن تماماً.



توصيل أحمال بتيارات كبيرة

أغلب المُتحكمات الدقيقة سواء القديمة أو حتى الحديثة نسبياً مثل ARM لا تستطيع أن تشغّل أحمال تسبّب في تيار أعلى من 50 مللي أمبير (أغلب المُتحكمات ARM يمكنها إمداد الأحمال بتيار ما بين 15 إلى 25 مللي أمبير). لحل هذه المشكلة يتم استخدام دوائر القيادة Driver circuits هذه الدوائر عبارة عن عناصر تعمل كمكّر للجهد أو التيار أو كليهما.

عادةً ما تبني هذه الدوائر باستخدام الترانزستور سواء BJT مثل NPN أو MOSFET ويكون الفارق الأساسي بينهما هو أقصى تيار يمكن أن يتحمله الترانزستور. حيث تتميز ترانزستورات MOSFET بتحمل تيارات قد تصل إلى 40 أمبير على عكس الدوائر NPN (or PNP) والتي لا تتحمل أكثر من 5 أمبير كأقصى تقدّر.

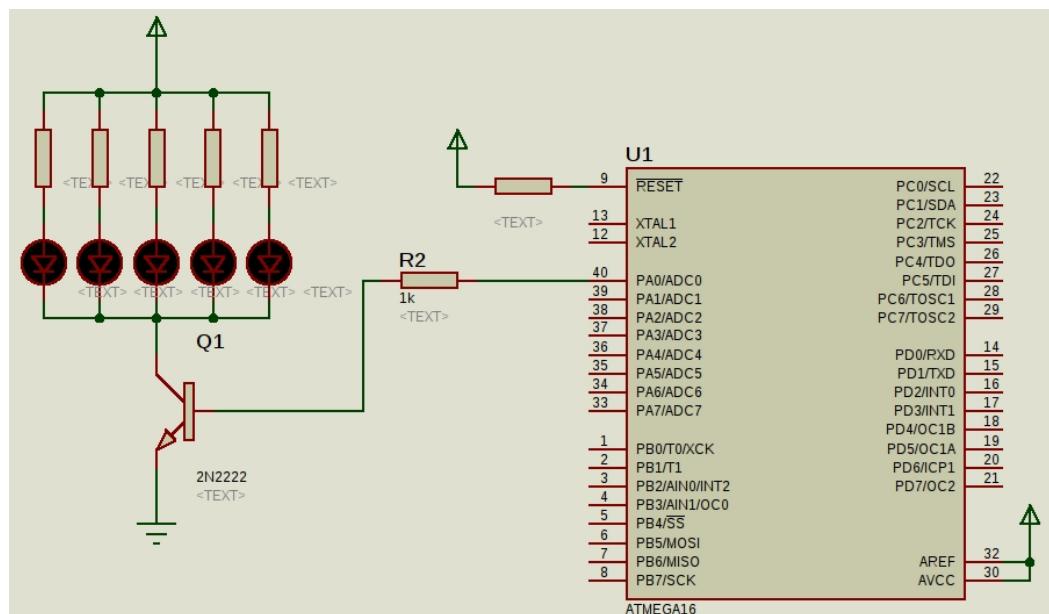
لأخذ مثال على هذه الدوائر: لنفترض أنك تريدين تشغيل 5 دايوّدات ضوئية ويجب أن تعمل معاً في نفس الوقت، سنجده أنه يتوفّر خياران لتشغيل هذه الدايوّدات، الأول هو استخدام 5 أطّراف من المُتحكم الدقيق وتوصيل كل دايوّد على طرف مستقل ولكن هذا الخيار يعدّ إهاراً لأطّراف المُتحكم.

الخيار الثاني هو توصيل الـ 5 دايوّدات كلها على طرف واحد باستخدام دائرة قيادة (وذلك لأنّ الدايوّدات الخمسة ستساهم في إجمالي 15 × 5 = 75 مللي أمبير، وهذا أكّبـر بكثير مما قد يتحمله المُتحكم على طرف واحد).

دائرة الترانزستور 2N2222

يُعدّ هذا الترانزستور أشهر أفراد عائلة الدوائر NPN وأكثرها توفّراً في الأسواق، كما يتميّز بالسعر المنخفض (يمكّنك شراء نحو 10 قطع منه بدولار واحد) ويستطيع تشغيل أحمال بتيار يصل إلى نصف أمبير (500 مللي)، بافتراض أنك لم تجده في السوق المحلي يمكنك شراء أحد البدائل مثل:

- مماثل لـ 2N2222 باستثناء أنه يتحمل 380 مللي أمبير فقط
- مماثل لـ 2N2222 باستثناء أنه يتحمل 100 مللي أمبير فقط





الدائرة بالأعلى تمثل طريقة توصيل المتحكم الدقيق مع الترانزستور، حيث يتم توصيل طرف المتحكم بقاعدة الترانزستور Base من خلال مقاومة يجب أن تكون قيمتها ما بين 150 أوم إلى 1 كيلو أوم.

يتم توصيل الديايدات الضوئية على طرف المجمع collector مع وضع مقاومة 300 أوم قبل كل دايد (الحملية الديايد) ويتم توصيل طرف المشع Emitter بالطرف الأرضي، نظرياً تستطيع هذه الدائرة أن تشعل حتى 30 دايد ضوئي (من النوع أحمر اللون) لكن كما أشرنا مسبقاً يستحسن أن يتم تشغيل أي عنصر إلكتروني بحد أقصى 85% من قدرته الكاملة وهذا يعني أن أقصى تيار للأحمال المتصلة على 22222n يجب أن يكون 400 ملي أمبير فقط.

بعد الانتهاء من توصيل هذه الدائرة يمكنك اختبارها عبر نفس المثال الأول Blinking led

ملاحظة: يتم استخدام مقاومة بين طرف المتحكم وقاعدة الترانزستور لأن أغلب الترانزستورات لا تتحمل تيار على طرف القاعدة أكبر من 10 ملي أمبير

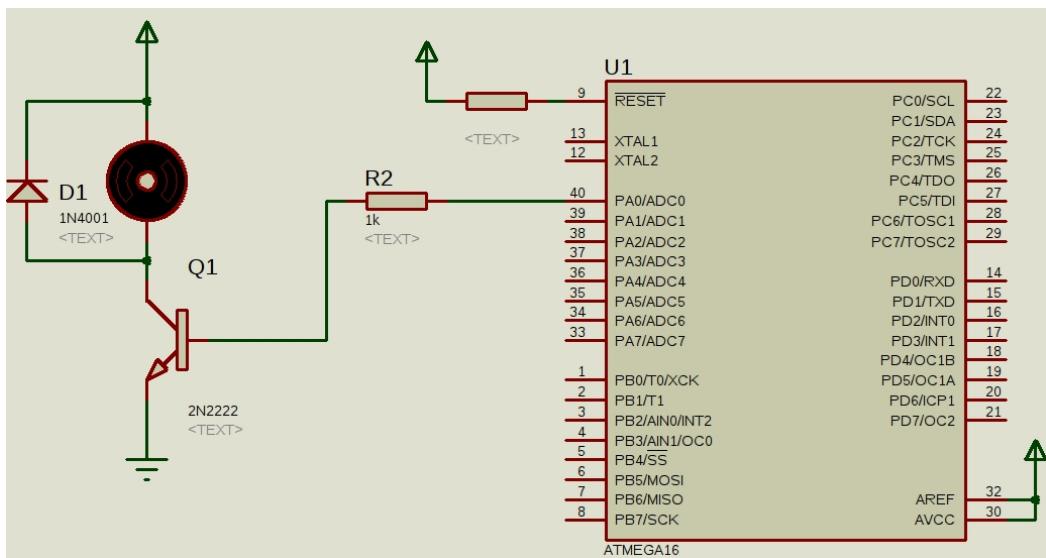


تشغيل المحركات DC

تعد المحركات (المواثير Motors) من أشهر العناصر الإلكترونوميكانيكية والتي عادةً ما نجدها في الأنظمة المدمجة الخاصة بالروبوتات، من أشهر هذه المحركات المحرك التيار المستمر DC motor (motor of the current). هذا المحرك من المكونات التي تستهلك الكثير من التيار الكهربائي لذا لا يمكن توصيله مباشرةً بالتحكم الدقيق ويجب أن نستخدم Driver circuit لتشغيله.

أبسط دائرة لتشغيل المحرك هي نفس الدائرة السابقة مع وجود بعض التعديلات البسيطة

ملاحظة: يحتوي برنامج بروتوك على محرك DC يسمى Active DC (Animated DC) وهذا هو الموجود في الصورة بالأعلى.

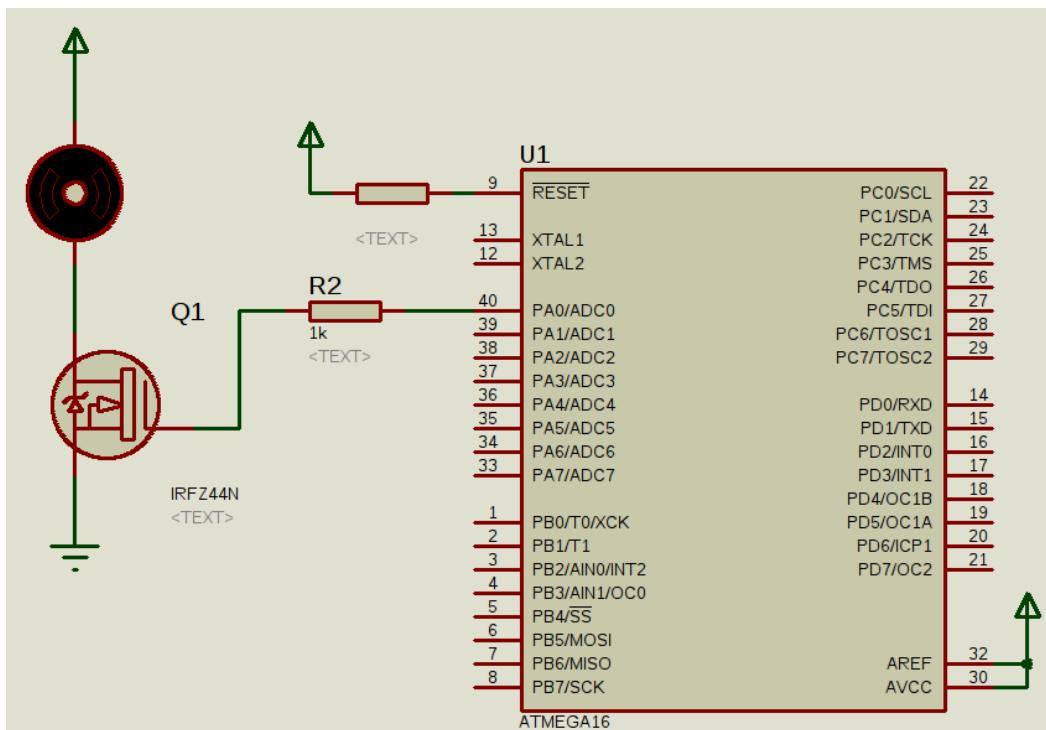


كما نرى من الصورة بالأعلى يتم توصيل المحرك في نفس مكان الدايمودات الضوئية باستثناء وجود دايمود n40011 متصل بطرف المحرك، يستخدم هذا الدايمود في الحفاظ على الترانزستور والمتحكم الدقيق من ظاهرة التيار المستحث العكسي.

هذه الظاهرة تحدث مع جميع الأحمال التي تتكون من ملف Coil مثل المحرك أو المُرجل Relay وذلك بسبب أن الملفات النحاسية تستطيع أن تخزن بعض الطاقة بداخليها وعند قطع الكهرباء عنها فإنها تقوم بتغريب هذه الطاقة المخزنة على هيئة تيار عكسي وقد يؤدي هذا التيار إلى تضرر الترانزستور والمتحكم الدقيق، ويتم استخدام الدايمود بصورة معكورة كما في الصورة السابقة لكي يمنع مرور هذه التيار من المحرك إلى الترانزستور.

ملاحظة: في حالة استخدام ترانزستور NPN مثل الـ 22222 يتطلب أن تستخدم مقاومة على طرف المشع (قبل أن يتصل بطرف الـ GND) وذلك للحفاظ على الترانزستور من السخونة والاحترق وهو ما يعرف باسم الـ thermal stability، حيث أن مرور التيار الخاص بتشغيل المحرك مباشرةً إلى الترانزستور NPN بدون وجود هذه المقاومة قد يؤدي إلى ارتفاع درجة حرارة الترانزستور مع الوقت ثم يحترق.

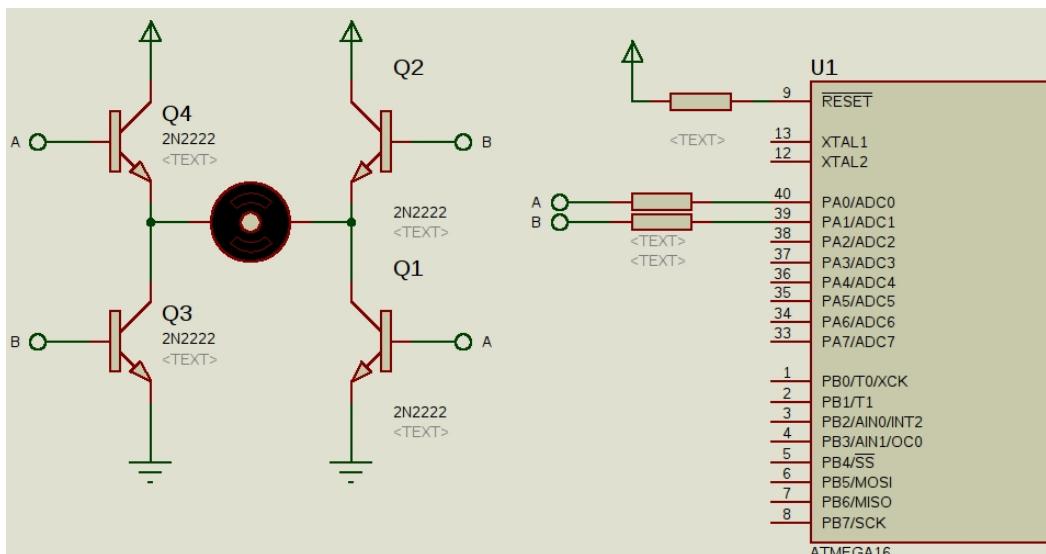
بالرغم أن الدائرة السابقة تصلح لتشغيل المحرك إلا أنه يفضل استخدام أحد الترانزستورات الـ MOSFET بدل الـ NPN حيث أنها تستطيع تحمل التيار الكهربائي العالي مثل الترانزستور IRZ44N (الذي يمكنه تحمل حتى 41 أمبير) كما هو موضح بالصورة التالية:



يمكنك تجربة هذه الدائرة بنفس الكود الخاص بالمثال الأول.

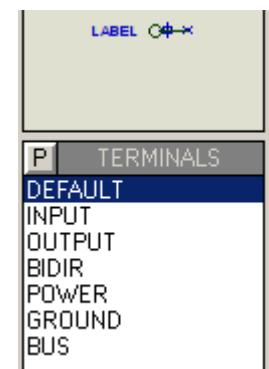
4.12 تشغيل المُحرّك في كلا الاتجاهين

جميع التجارب السابقة لتشغيل المحرك دائرة القيادة المعتمدة على الترانزستور كانت تعمل على تشغيل المحرك في اتجاه واحد فقط، ماذا إذا أردنا تشغيل المحرك في اتجاهين مختلفين؟ يمكننا ذلك باستخدام ما يعرف باسم H-Bridge وهي قطرة مكونة من 4 ترانزستور على شكل الحرف H وينصل المحرك بوسط هذه القطرة مثل الشكل التالي:



ملاحظة: الدوائر التي تحتوي على حرف A,B تسمى Default Terminal وهي نقاط توصيل تستخدم لتوصيل المكونات ببعضها البعض بصورة سهلة بدلاً من الأسلك المتداخلة ويمكنك الوصول إليها عبر الضغط على قائمة Terminal ثم اختيار Default.

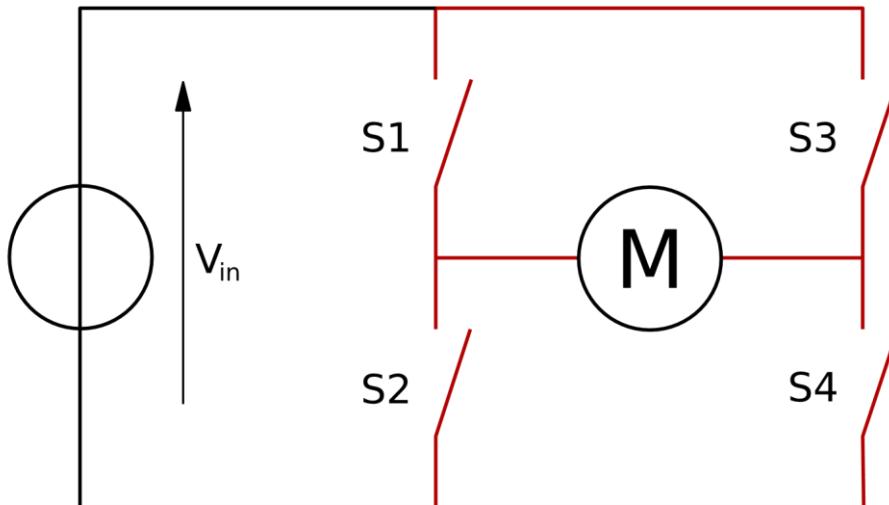
لتوصيل أكثر من نقطة ببعضها البعض كل ما عليك فعله هو الضغط على هذه الدائرة وتسميتها بحرف معين مثل A,B,C,D وستقوم هذه الدوائر بتوصيل جميع الدوائر المسماة بنفس الاسم ببعضها البعض.



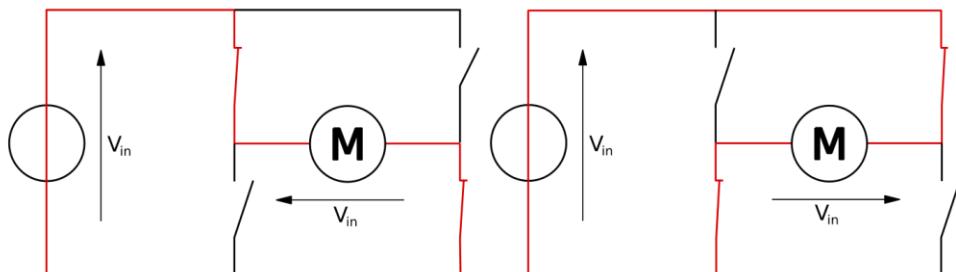


مبدأ عمل قنطرة H

تعمل الـ H-bridge مثل 4 مفاتيح، كما هو موضح في الصورة التالية



يُعمل كل مفاتيح مع بعضهما البعض، فإذاً أن يتم إغلاق المفتاح S1 و S4 بحيث يمر التيار الكهربائي من الطرف الأيسر إلى الطرف الأيمن للمحرك (بذلك يدور المحرك مع عقارب الساعة)، أو يتم إغلاق المفاتيح S2 و S3 بحيث يمر التيار الكهربائي من الطرف الأيمن إلى الطرف الأيسر وبالتالي يدور المحرك عكس عقارب الساعة.



البرنامج

لتشغيل هذه القنطرة يتم استخدام طرقين مثل PA0 و PA1 حيث يتصل كل طرف بعده 2 من الترانزستورات كما هو موضح في الصورة الأولى، ويتم التحكم باتجاه دوران المحرك عن طريق تشغيل الطرف PA0 وإطفاء PA1 أو العكس (لتغيير اتجاه دوران المحرك).

```
#define F_CPU 1000000UL
#include <avr/io.h>
```



```
#include <util/delay.h>

int main(void)
{
    DDRA = 0b11111111;

    while(1)
    {
        PORTA = 0b00000001; // تشغيل المحرك مع عقارب الساعة
        _delay_ms(2000); // انتظر لمدة ثانيتين
        PORTA = 0b00000010; // تشغيل المحرك عكس عقارب الساعة
        _delay_ms(2000); // انتظر لمدة ثانيتين
        PORTA = 0b00000000; // إطفاء جميع الترانزستور وإيقاف المحرك
        _delay_ms(2000); // انتظر لمدة ثانيتين
    }
    return 0;
}
```



صفحة جديدة



5. قواعد لغة السي للأنظمة المدمجة

الفصل
القواعد
الشهير للغة
المعيارية
بشكل كبير
الأنظمة
تمييز الصيغ
بإمكانية
مختلف
الدقة طالما
الخاص بها
السي.

✓ أنواع
البيانات في
الأنظمة
المدمجة
Data-
types
✓ العمليات
الحسابية



في هذا
سنتعلم بعض
والصيغ
السي
والمستخدمة
في تطوير
المدمجة.
المعيارية
تطبيقاتها على
المتحكمات
أن المترجم
يدعم لغة

For Embedded Systems

واختصاراتها Arithmetic operations

العمليات المنطقية واحتياطاتها Logic operations

✓

عمليات الأزاحة

✓

التلاعب بالبتات

✓

5.1 أنواع البيانات في الأنظمة المدمجة Data-types

في عالم الأنظمة المدمجة هناك معيار مختلف قليلاً لتعريف البيانات مثل المتغيرات والثوابت. حيث نجد أن معظم المتحكمات الدقيقة خاصة من فئة Bit-8 تمتلك قدر محدود جداً من الذاكرة سواء الـ ROM أو الـ RAM (بعض المتحكمات قد يتمتلك 128 بait فقط من الـ RAM). مما يتطلب أسلوب فعال لكتابة أكواد تحقق الهدف المطلوب بأقل استهلاك للذاكرة.

مثلاً في حالة كتابة برنامج بلغة السي على الحاسوب الآلي التقليدي والذي يمتلك معالج وذاكرة ضخمة (تقدر بـ 32 بت) وكلاهما مخصص للتعامل مع البيانات 64 أو 32 بت، سنجد أن معظم المتغيرات الرقمية تكون int أو double أو float أما في حالة الأنظمة المدمجة يعد استخدام هذه الأنواع "اهدار للذاكرة" لذا سنتعلم في هذا الجزء كيفية برمجة أنواع البيانات المختلفة بصورة محسنة خصيصاً لذاكرة المتحكمات الدقيقة.

البيانات الرقمية

تعتبر الأرقام هي أكثر أنواع البيانات استخداماً في عالم الأنظمة المدمجة فهي تعبر عن قيم المتغيرات مثل قراءات الحساسات (حرارة، ضغط، رطوبة، ضوء، .. الخ) أو قيم المخارج مثل قيمة أي PORT أو سرعة Motor أو توقيت زمني .. الخ. وتعتبر الأرقام هي أكثر ما يستهلك الذاكرة (خاصة الـ SRAM) لذا سيكون لها نصيب كبير من الشرح في هذا



الفصل.

أنواع البيانات التقليدية

هذه القائمة تمثل أشهر أنواع البيانات المستخدمة في تعريف المتغيرات في لغة السي مع حجم الذاكرة الذي تستهلكه كل منها. مع العلم أنه يمكنك برمجة جميع الأنظمة المدمجة بها.

أنواع البيانات التي تستخدم في حفظ الأرقام والموجبة والسلبية تسمى **signed numbers** أما إذا وضعنا قبلها كلمة **unsigned** فإنها تصلح للتعامل مع الأرقام **الموجبة فقط** ولكن بضعف مساحة التخزين

نوع البيانات	استهلاك الذاكرة (عدد Bytes)	أقصى قيمة يمكن وضعها داخل المتغير
int	2-bytes (16 bit)	-32,768 → +32,767
unsigned int	2-bytes (16 bit)	65,535
float	4-bytes (32 bit)	1.2E-38 to 3.4E+38
double	8-bytes (32 bit)	2.3E-308 to 1.7E+308
long	4-bytes (32 bit)	-2,147,483,648 → +2,147,483,647
unsigned long	4-bytes (32 bit)	4,294,967,295
long long	8-bytes (64 bit)	- 9.223372037×10 ¹⁸ + 9.223372037×10 ¹⁸
unsigned long long	8-bytes (64 bit)	1.844674407×10 ¹⁹

ملاحظة: نوع البيانات **float** و **double** دائمًا يكونا من نوع **Signed** ولا يمكن استخدام العلامة **unsigned** معهما

كما نلاحظ من الجدول السابق أن جميع أنواع البيانات الشهيرة تستهلك مساحة تبدأ من 2 بait حتى 8 بait وتعتبر هذه المساحة كبيرة نسبياً في عالم الأنظمة المدمجة. فمثلاً قيمة أي مسجل مثل PORTx أو PINx أو DDRx لن تزيد أبداً عن 8 بت.

والآن لنتخيل أننا نحتاج متغير اسمه **ButtonStatus** من نوع **int** وسيستخدم هذا المتغير في حفظ قراءة أطراف البورت B وبما أن المُسجل PINB من نوع 8 بت إذا كل ما يتطلبه هو متغير بسعة 8 بت فقط ولا داعي أبداً لاستخدام متغير بمساحة 16 بت. لذا فجميع أنواع البيانات السابقة تعتبر اهدر للذاكرة.

لحل هذه المشكلة يتم استخدام أنواع البيانات المعيارية ANSI - C99 - C - C99 وهي صورة محسنة لأنواع البيانات وتمكننا من اختيار قيمة المتغيرات بالدقة والطول المطلوب دون أي اهدر للذاكرة.

Signed	أقصى قيمة	Unsigned	أقصى قيمة
int8_t	-128 → +127	uint8_t	0 → 255
int16_t	-32,768 → +32,767	uint16_t	0 → 65,535
int32_t	-2,147,483,648 → +2,147,483,647	uint32_t	0 → 4,294,967,295
int64_t	- 9.223372037×10 ¹⁸	uint64_t	1.844674407×10 ¹⁹



$$+ 9.223372037 \times 10^{18}$$

يستهلك كل متغير من القائمة السابقة مساحة = الرقم المكتوب بعد كلمة **int** أو **uint** **int8_t** فمثلاً من نوع **int** ويصلح لتخزين الأرقام **الموجبة والسلبية** في مساحة تخزينية = 8 بت فقط (1 بait). بينما **uint16_t** يعني أن هذا المتغير من نوع **unsigned int** ويصلح لتخزين الأرقام الموجبة فقط بمساحة تصل إلى 16 بت (2 بait). والآن لنأخذ مجموعة من الأمثلة:

ما هو نوع المتغير المطلوب لقراءة أو الكتابة في أي بورت (مثل **A, B, C**)؟

- بما أن جميع البورتات يتم قراءتها من خلال المُسجل **PINx** والذي يكون 8 بت وكذلك يتم كتابة في أي بورت باستخدام المُسجل **PORTx** والذي يعتبر 8 بت أيضاً إذا أفضل نوع بيانات هو **uint8_t** حيث يمكنه تخزين الأرقام من 0 إلى 255.

ما هو المتغير المناسب لتخزين عدد رقمي من صفر إلى 10,000؟

- بما أن الرقم لا يحتوي على أرقام سالبة وأقصى رقم مطلوب هو 10000 وهذا الرقم أكبر من 255 لذا لا يمكن استخدام متغير بمساحة 8 بت و يجب أن نستخدم متغير بمساحة 16 بت والذي يستطيع التعامل مع ارقام تصل إلى 65,535. إذا أفضل نوع بيانات هو **uint16_t**.

ما هو المتغير المناسب لتخزين درجة حرارة تتراوح بين سالب 50 إلى موجب 100؟

- هناك إجابتين لهذا السؤال، الأولى: في حالة أننا نريد تخزين أرقام صحيحة فقط مثل 20 أو 50 أو أي رقم بدون كسر عشري فسنجد أن أفضل نوع بيانات هو **int8_t** حيث يستطيع هذا النوع من البيانات أن يخزن أي قيمة تتراوح بين -128 إلى 127+ وستهلك 1 بait فقط من مساحة الذاكرة.
- الثانية: في حالة وجود كسر عشري مثل أن 5.1 أو 30.5 يجب أن نستخدم نوع البيانات **float** مع العلم أنه في المقابل سيتطلب مساحة ذاكرة = 4 بait (32 بت).

السؤال الأخير يدفعنا لنقطة هامة جداً وتسميـ الـ **Variable precision** - دقة المتغير - بعض المتغيرات يجب أن يتم تخزينها بدقة عالية جداً. والبعض الآخر لا يتطلب هذه الدقة فمثلاً إذا طلب منك أن تصنع ساعة رقمية وبها حساس لعرض درجة حرارة الغرفة، سنجد أن المتغير المناسب لتخزين قيم هذا الحساس هو **int8_t** وذلك لأن درجة الحرارة المترقبة لأي غرفة لن تزيد عن 45 أو تقل عن -5 (حتى أقصى بلاد العالم بروادة تكون درجة حرارة المنازل معزولة عن درجة الهواء الخارجي بفارق 10 درجة على الأقل) كما أن الشخص الذي سيستخدم هذه الساعة الرقمية لن يهتم بعرض درجة الحرارة بدقة 25.20 وسيكتفي فقط بمعرفة الرقم الصحيح (مثل 25 درجة). ولكن إذا طلب منك تصميم مقياس حرارة طبي مثل المستخدم في المستشفيات لقياس درجة حرارة المرضى فيجب أن تكون درجة الحرارة محددة بدقة مثل 36.25 لذا يجب أن يتم استخدام متغير من نوع **float** لتخزين و معالجة هذه القيم.

أمثلة برمجية

قم بتوصيل 8 دايوهات ضوئية على أطراف البورت A ثم اكتب برنامج يقوم بإخراج القيم من 0 إلى 255 على البورت مع وضع تأخير زمني بين كل قيمة 100 مللي ثانية.

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <avr/delay.h>
```

```
int main(void)
```

```
{
```

```
    uint8_t counter;
```

متغير بمساحة 8 بت لتخزين قيمة العداد //



```
DDRA = 0b11111111;           // ضبط كل أطراف البورت لعمل كخرج
                                // عداد من 0
for (counter = 0; counter <= 255; counter++)           // عداد من 0
255
{
    PORTA = counter;
    _delay_ms(100);
}
return 0;
```



صور مختلفة لكتابة المتغيرات الرقمية

في لغة السي يمكن كتابة قيم المتغيرات الرقمية بطرق متعددة. فمثلاً في الأمثلة السابقة قمنا باستخدام الصيغة الرقمية الثنائية Binary مثل:

```
uint8_t variable = 0b00000011;
```

أيضاً يمكن كتابة المتغيرات بالصيغة العشرية Decimal وهي نفس صيغة الأرقام التي نستخدمها في حياتنا اليومية فمثلاً الرقم 10 يساوي رقم 3 Decimal وبإمكان كتابة نفس الأمر السابق بالصورة التالية:

```
uint8_t variable = 3;
```

ويمكن استخدام الصيغة الستة-عشرية Hexadecimal. فمثلاً لنفترض أنتا نريد تعين المتغير variable بقيمة 120 والتي تساوي رقم 120 بالصيغة العشرية

```
uint8_t variable = 0b11110000; // Binary
uint8_t variable = 120; // Decimal
uint8_t variable = 0xF0; // Hexadecimal
```

مرفق مع الكتاب ملحق خاص بنظام الأعداد وتحويل قيم الأرقام من وإلى الصيغ الثنائية والعشرية والستة-عشرية.
أيضاً لاحظ أن القيم الستة-عشرية تبدأ بـ **x0** بينما الصيغة الثنائية تبدأ بـ **0b**.

مراجع إضافية

<http://www.mjma3.com/programming/basics.html>

فيديو: تحويل الأعداد من النظام العشري إلى الثنائي

<https://www.youtube.com/watch?v=-yrwpEuRjl0>

فيديو: تحويل الأعداد من النظام الثنائي إلى العشري

https://www.youtube.com/watch?v=ILbc5_JpTWI

فيديو: تحويل الأعداد من النظام الثنائي إلى الست-عشرى

<https://www.youtube.com/watch?v=B33Iof0bUKg>



5.2 العمليات الحسابية Arithmetic Operations

تستطيع مترجمات لغة السي المعايير المخصصة للمتحكمات الدقيقة أن تتعامل مع نفس أوامر العمليات الحسابية التي يمكن تنفيذها على الحواسيب مثل عمليات الجمع والطرح والضرب والقسمة. لذا نأخذ بعض الأمثلة:

تعريف 3 متغيرات

عملية جمع رقمين ووضع الناتج في المتغير **X** //

عملية طرح رقم من متغير ووضع الناتج في المتغير **Y** //

عملية ضرب متغيرين ووضع الناتجة في المتغير **Z** //

أيضاً يمكن تنفيذ العمليات الحسابية على قيمة المتغير نفسه. فمثلاً قيمة المتغير **x** في المثال السابق = 11 والآن نريد أن نزيد عليها 4 لتصبح 15. يمكن تنفيذ ذلك بالطريقة التالية

X = X+4; // الآن قيمة المتغير تساوي 15

الأمر السابق يعني: قم بإضافة الرقم 4 إلى القيمة الموجودة بالفعل في المتغير **X** ثم قم بكتابة النتيجة مرة أخرى داخل المتغير **X** وبذلك تصبح قيمة 15. مع العلم أنه يمكن إجراء نفس الطريقة مع جميع العمليات الحسابية

X = X - 2; // أطرح الرقم 2 من المتغير

X = X * 4; // اضرب قيمة المتغير في 4

الصور المختصرة

يمكن إجراء العمليات الحسابية على المتغيرات بصورة مختصرة. فمثلاً إذا أردنا جميع الرقم 4 على قيمة المتغير **X** فيمكن ذلك بأسلوب مختصر عبر إضافة علامة الجمع قبل إشارة =

X += 4; // **X = X+4** اختصار الأمر

أيضاً يمكن تطبيق نفس الاختصار على أي عملية حسابية

X -= 4; // **X = X - 4** اختصار الأمر

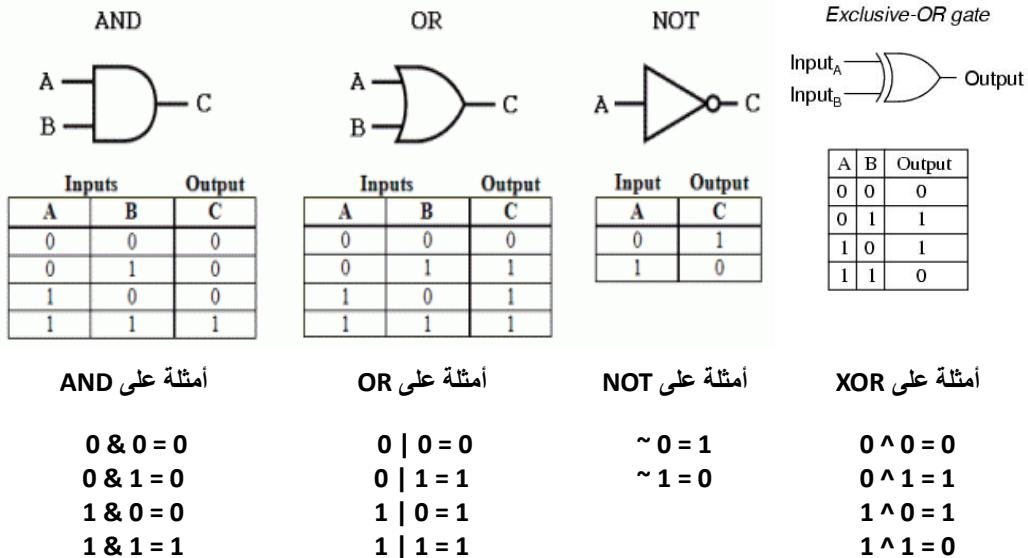
X *= 4; // **X = X * 4** اختصار الأمر

X /= 2; // **X = X / 2** اختصار الأمر

5.3 العمليات المنطقية Logic Operation

الـ Logic operation مجموعة من الأوامر المنطقية مثل AND, OR, XOR, Invert, Shift هذه الأوامر تساعدنا على القيام ببعض التحكم المتقدم سواء على المتغيرات أو إعدادات المتتحكم أو أطراف المتتحكم (الدخل أو الخرج). جميع العمليات المنطقية (تسمى أيضاً الدوال المنطقية Logic Function) تمثل نفس البوابات المنطقية المستخدمة في الدوائر الإلكترونية لكن يتم تطبيقها بصورة برمجية. الجدول التالي يوضح قائمة بهذه العمليات والرموز المستخدمة في لغة السي لتنفيذ كل وحدة منهم.

Logic Function name	Logic Function in "ANSI C"
OR	
AND	&
NOT	~
XOR	^
Shift Left	<<
Shift Right	>>



أمثلة على AND

$$\begin{aligned}0 &\& 0 = 0 \\0 &\& 1 = 0 \\1 &\& 0 = 0 \\1 &\& 1 = 1\end{aligned}$$

أمثلة على OR

$$\begin{aligned}0 &\mid 0 = 0 \\0 &\mid 1 = 1 \\1 &\mid 0 = 1 \\1 &\mid 1 = 1\end{aligned}$$

أمثلة على NOT

$$\begin{aligned}\sim 0 &= 1 \\\sim 1 &= 0\end{aligned}$$

أمثلة على XOR

$$\begin{aligned}0 \wedge 0 &= 0 \\0 \wedge 1 &= 1 \\1 \wedge 0 &= 1 \\1 \wedge 1 &= 0\end{aligned}$$

لنفهم هذه العمليات بصورة أفضل نأخذ مجموعة من الأمثلة البرمجية.

استخدام NOT - يُستخدم الأمر NOT “invert” في عكس جميع البتات داخل أي متغير أو أي مُسجّل فمثلاً إذا كان لدينا المتغير Z كالتالي:

Example 1:

```
unit8_t Z = 0b00000000;
Z = ~Z; // invert all Z bits (Z = 0b11111111)
```

تطبيق الأمر الأخير يعني: قم بعكس جميع البتات الموجودة في المتغير X ثم ضع القيمة الجديدة داخل المتغير X وبذلك تصبح القيمة = b11111110

Example 2:

```
unit8_t Z = 0b11110000;
Z = ~Z; // invert all Z bits (Z = 0b00001111)
```

Example 3:

```
unit8_t Z = 0b00110011;
Z = ~Z; // invert all Z bits (Z = 0b11001100)
```

استخدام OR - لنفترض أن لدينا متغير 8 بت xNumber وقيمتها = 12 (بالنظام العشري) أو 0b000110000 بالنظام الرقمي كما هو موضح:

Example 4:

```
uint8_t xNumber = 0b00011000;
xNumber = xNumber | 0b00000001; // 0b000011000 OR 0b00000001
```

الأمر السابق يعني قم بتنفيذ العملية المنطقية OR مع محتوى المتغير xNumber ثم ضع النتيجة داخل المتغير xNumber مما سيجعل قيمة المتغير تصبح 0b000110010. وذلك لأن الأمر OR يقوم بعمل OR operation على كل بت بين الرقمين كالتالي:



قيمة xNumber الأصلية	0	0	1	1	0	0	0	0
الرقم الثاني	0	0	0	0	0	0	0	1
نتيجة الـ OR	0	0	1	1	0	0	0	1

Example 5:

```
xNumber = 0b11111000;
xNumber = xNumber | 0b00000011;
```

عند تنفيذ الأمر OR ستكون قيمة المتغير = 0b111110110 كما هو موضح في الجدول التالي:

قيمة xNumber الأصلية	1	1	1	1	1	0	0	0
الرقم الثاني	0	0	0	0	0	0	1	1
نتيجة الـ OR	1	1	1	1	1	0	1	1

استخدام AND – تعمل عملية الـ AND مثل ضرب رقمين. وعند تطبيقها بين المتغيرات يتم عمل AND بين كل بت في المتغير الأول وما يقابلها في المتغير الثاني

Example 6:

```
;xNumber = 0b11111000
// 0b11111000 AND 0b10000001      xNumber = xNumber & 0b10000001;
النتيجة ستكون 0b10000000 وذلك لأن نتيجة عمل AND مع أي بت تحتوي على صفر = صفر لذا نجد البت الأخير
فقط هي التي ستظل 1 لأن 1 & 1 = 1
```

قيمة xNumber الأصلية	1	1	1	1	1	0	0	0
الرقم الثاني	1	0	0	0	0	0	0	1
نتيجة الـ AND	1	0	0	0	0	0	0	0

اختصار العمليات المنطقية

يمكن اختصار العمليات المنطقية مثل العمليات الحسابية وذلك عبر وضع الأمر المنطقي قبل علامة = فمثلاً يمكن إجراء الـ logic operations كالتالي:

```
xNumber |= 0b00000001;           // xNumber = xNumber |
0b00000001;
xNumber &= 0b00000001;           // xNumber = xNumber &
0b00000001;
```



```
xNumber ^= 0b00000001;           // xNumber = xNumber ^  
0b00000001;
```

ملاحظة: يمكن اختصار جميع العمليات المنطقية ماعدا عملية (العكس Invert) والتي تكتب فقط بالصورة التالية:
 $xNumber = \sim xNumber$



5.4 عمليات الإزاحة Shift operations

تعرف عمليات الإزاحة بأنها تحريك البنايات إلى اليمين أو اليسار داخل متغير أو مسجل أو أي قيمة رقمية. تعد هذه العمليات من أهم الأوامر التي تستخدم في برمجة المعالجات والفتحات الدقيقة كما سنرى.

مثال: لنفرض أن لدينا الرقم 0b000000010 لنقم بتطبيق عمليات الإزاحة لجهة اليسار.

الرقم الأصلي	0b00000001
إزاحة بمقدار 1 بت جهة اليسار <<	0b00000010
إزاحة بمقدار 2 بت جهة اليسار <<	0b00000100
إزاحة بمقدار 3 بت جهة اليسار <<	0b00001000

مثال: لنفرض أن لدينا الرقم 0b000001010 لنقم بتطبيق نفس العمليات السابقة عليه.

الرقم الأصلي	0b00000101
إزاحة بمقدار 1 بت جهة اليسار <<	0b00001010
إزاحة بمقدار 2 بت جهة اليسار <<	0b00010100
إزاحة بمقدار 3 بت جهة اليسار <<	0b00101000

مثال: لنفرض أن لدينا 0b011000000 لنقم بتطبيق عمليات الإزاحة لجهة اليمين.

الرقم الأصلي	0b01100000
>> إزاحة بمقدار 1 بت جهة اليمين	0b00110000
>> إزاحة بمقدار 1 بت جهة اليمين	0b00011000
>> إزاحة بمقدار 1 بت جهة اليمين	0b000011000

تدعم لغة السي الأوامر البرمجية لعمل الإزاحة لجهة اليمين أو اليسار وتكتب كالتالي:

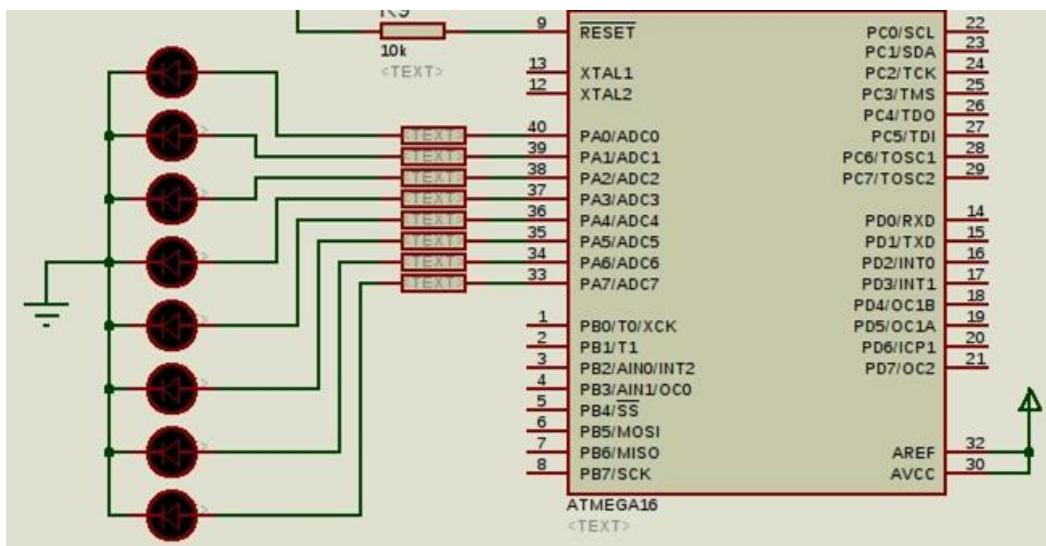
الإزاحة لجهة اليسار:

(قيمة الإزاحة << الرقم)

الإزاحة لجهة اليمين:

(قيمة الإزاحة >> الرقم)

المثال الأول: قم بتوصيل 8 دايودات ضوئية على البورت A ثم جرب البرنامج التالي:



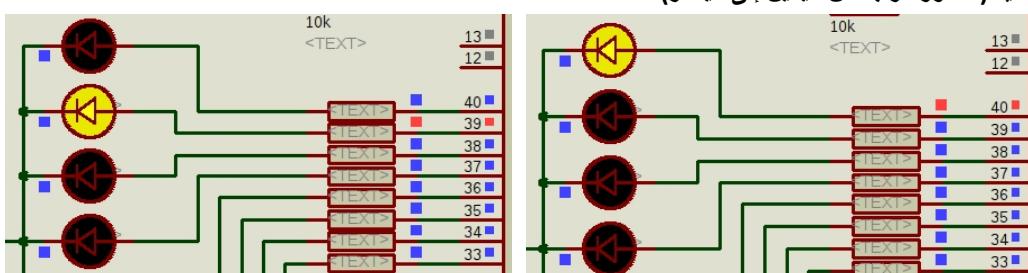
```

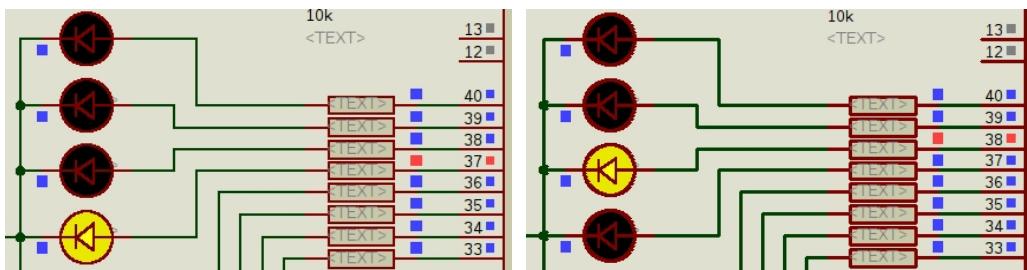
int main(void)
{
    uint8_t counter;
    DDRA = 0xff;

    while(1)
    {
        for(counter = 0; counter <= 7; counter++)
        {
            PORTA = (1 << counter);
            _delay_ms(500);
        }
    }
    return 0;
}

```

نتيجة تنفيذ الكود ان الدايوارات الضوئية ستبدأ بالاضاءة واحدة تلو الأخرى بداية من PA0 إلى PA7 بالترتيب كما في الصور التالية (الصور مرتبة من اليمين إلى اليسار):





شرح الكود

في البداية تم تعريف متغير يسمى **counter** حيث ستستخدم هذا المتغير في زيادة قيمة الإزاحة بصورة تلقائية. أيضاً تم تعريف جميع أطراف البورت A لعمل كخرج.

```
uint8_t counter;
DDRA = 0xff;
```

داخل الـ **loop** استخدمنا دالة التكرار **for** وذلك لزيادة قيمة المتغير **counter** بصورة تلقائية من 0 إلى 7.
for(counter = 0; counter <= 7; counter++)
والآن يأتي أمر المحاداة. حيث تم استخدام البورت A لعرض كيف تتحرك البت 1 لجهة اليسار مع تأخير زمني نصف ثانية وهذا ما يسبب أن تضيء الدايرودات الضوئية بالترتيب واحدة تلو الأخرى.

```
PORTA = (1 << counter);
_delay_ms(500);
```

لاحظ أن الرقم 1 (المكتوب بالصيغة العشرية) الموجود في العبارة **PORTA = (1 << counter)** يوازي بالضبط **(0b00000001 << counter)**

بسبب الدالة **for** نجد أنه عندما تكون قيمة **counter = 1** فهذا يجعل الأمر السابق يساوي **PORTA = (0b00000001 << 1);**

وعندما تكون قيمة **counter = 1** فهذا يوازي الأمر

```
PORTA = (0b00000001 << 2);
```

وعندما تكون قيمة **counter = 3** فهذا يوازي الأمر

```
PORTA = (0b00000001 << 3);
```

وهكذا ... وتنstemr هذه الدورة إلى ما لا نهاية.

5.5 التحكم على مستوى البت الواحدة Single Bit

في الفصل السابق قمنا بالتعامل مع بعض المسجلات مثل **PINx**, **PORTx**, **DDRx** وذلك بكتابة بعض القيم الرقمية داخل هذه المسجلات. جميع الأمثلة السابقة اشتهرت في أمر **ham** وهو كتابة جميع قيم البتات داخل كل مسجل في نفس الوقت. فمثلاً عندما نقوم بضبط أول 3 أطراف للبورت A فلنكتب **DDRA = 0b00000111** ما يجعل الأطراف الثلاث الأولى خرج والأطراف الباقيه تعمل كدخل.

لكن ماذا إذا أردنا تعديل طرف واحد (ولتكن الطرف الأول فقط **PA0**) وفي نفس الوقت لا نريد أن نغير قيمة أو محتوى أي بٍت آخر في المسجل؟ **DDRA**؟

يمكننا ذلك بسهولة عبر دمج مجموعة من العمليات المنطقية وعمليات الإزاحة. حيث تتوفّر صيغ برمجية معينة تمكننا من وضع 1 أو 0 داخل بٍت محددة في أي متغير أو مسجل. كما يمكننا أن نعكس بٍت معينه أو نعزلها عن باقي البتات.



كتابة 1 داخل أي بت Set Bit

تعرف الصيغة البرمجية Set Bit أنها وضع 1 داخل قيمة أي بت. بالعودة للسؤال السابق لنفترض أننا نريد ضبط الطرف PA0 فقط ليعمل كخرج بغض النظر عن باقي أطراف البورت A. يمكن ذلك بسهولة عبر دمج الأمر left shift مع OR مع تكون الصيغة العامة

Register |= (1 << bitName);

يتم استبدال كلمة Register بالمسجل المطلوب ويتم استبدال كلمة bitName باسم البت المطلوب تغيير قيمتها. فمثلاً بتطبيق هذه الصيغة على المُسجِل DDRA و الطرف PA0 نكتب:

DDRA |= (1<< PA0); // set PA0 to work as output

أيضاً يمكننا استخدام نفس الصيغة لتشغيل نفس البت من المُسجِل PORTA

PORTA |= (1<<PA0); // turn on PA0

كذلك يمكننا استخدام العملية OR لدمج أكثر من صيغة من النوع السابق في أمر واحد، فمثلاً لنفترض أننا نريد ضبط الطرف PA1 و PA2 ليعملان كخرج، يمكننا إما أن نكتب:

DDRA |= (1<<PA1);

DDRA |= (1<<PA2);

أو يمكن اختصار الأمرين في جملة واحدة فقط كالتالي:

DDRA |= (1<<PA1) | (1<<PA2);

كتابة 0 داخل أي بت Reset Bit

عملية الـ Reset Bit هي عكس عملية Set Bit وتعني وضع صفر داخل أي بت مطلوبة وتنتمي عبر دمج 3 أوامر AND – left shift – invert و تكون الصيغة القياسية كالتالي:

Register &= ~ (1 << bitName);

مثال: لنفترض أننا نريد جعل الطرف PC3 يعمل كدخل بغض النظر عن باقي أطراف البورت

DDRC &= ~(1<<PC3);

أيضاً يمكن استخدام نفس الصيغة في إلغاء تشغيل أي طرف، فمثلاً إذا أردنا تشغيل دايوود ضوئي على البت PA0 لمدة ثانية واطفاؤه لمدة ثانية فيمكننا أن نكتب:

```
int main() {
    DDRA |= (1<<PA0); // set 1 in PA0 (output)
    while (1)
    {
        PORTA |= (1<<PA0); // turn on led (set PA0)
        delay_ms(1000);

        PORTA &= ~(1<<PA0); // turn off led (reset PA0)
        delay_ms(1000);
    }
    return 0;
}
```



عكس بت محددة

تعد الصيغة البرمجة Toggle Bit من أفضل الصيغ المستخدمة في البرمجة وتعني أنه في حالة أن البت المطلوبة = 1 قم بعكسها إلى 0 وإذا كانت تساوي 0 قم بعكسها إلى 1. وتكون الصيغة الفياسية كالتالي:

```
Register ^= (1 << BitName);
```

باستخدام هذه الصيغة يمكننا اختصار المثال - led blinking بأمرین فقط (بدلاً من 4 أوامر).

```
while (1)
{
    PORTA ^= (1 << PA0);
    _delay_ms(1000);
}
```

تعد مجموعة الصيغ البرمجية السابقة من أهم الأوامر التي قد تستخدمها في برمجة النظم المدمجة خاصة أنها مكتوبة بلغة السي المعيارية مما يجعلها طريقة "معيارية" لعمل reset أو set لأي بت مهما أختلف المتحكم الدقيق أو الشركة المنتجة (طالماً أن المترجم الخاص بهذا المتحكم يدعم لغة السي المعيارية).

5.6 القراءة من بت واحدة

في الفصل السابق شاهدنا بعض الأمثلة عن قراءة مفتاح الكتروني (switch) push button وذلك عبر قراءة بورت كامل من خلال المُسجل PINx. الطريقة السابقة كانت تقرأ جميع بباتات البورت PINx وتقارنها برقم مثل:

```
if (PINB == 0b00000001)
    {PORTA = 0b00000001;}
else
    {PORTA = 0b00000000;}
```

لكن هناك صيغة أفضل في حالة الرغبة بقراءة طرف واحد فقط وهي كالتالي:

Reister & (1<<bitName)

يمكن تطبيق نفس الصيغة على الكود السابق ليصبح كالتالي:

```
if (PINB & (1 << PA0)) //1
    {PORTA = 0b00000001;}
```



صفحة جديدة



6. الفيوزات، الحماية، الطاقة وسرعة التشغيل



هذا الفصل يشرح الإعدادات المتقدمة لمحكمات AVR مثل مفهوم الفيوزات ووظائفها المختلفة مثل تغير سرعة التشغيل Clock Rate واستهلاك الطاقة، حماية البرامج الموجودة على المُتحكم من السرقة أو التعديل وتشغيل بعض الخصائص المتقدمة الأخرى.

الفيوزات Fuses لـ المُتحكم ATmega16	✓
تغير سرعة (تردد) المُتحكمات	✓
بتات الإغلاق والحماية Lockbits	✓
برمجة الفيوزات	✓
استهلاك الطاقة والعمل على البطاريات	✓



Fuses & Lockbits 6.1

في جميع المُتحكمات الدقيقة يجب أن تمتلك طريقة ما للتحكم في بعض الإعدادات بصورة دائمة لا تتغير مع انقطاع الكهرباء عن المُتحكم الدقيق، فمثلاً مُتحكمات الـ AVR تأتي بصورة افتراضية من المصنع تعمل بتردد 1 ميجاهرتز وقد تحتاج أن تغير هذه السرعة ليعمل المُتحكم دائمًا بسرعة 16 ميجاهرتز، مثل هذه الإعدادات يجب أن تتم مرة واحدة ولا تتغير في المستقبل.. هنا يأتي دور الفيوزات Fuses.

الفيوزات هي وحدات ذاكرة (بيتات Bits) يتم برمجتها بصورة مستدامة، هذا يعني أن محتواها لا يتغير بإعادة تشغيل المُتحكم أو انقطاع الكهرباء أو حتى برمجة الذاكرة الرئيسية (Flash Memory)، هذه الفيوزات تختلف بمجموعة خاصة من الإعدادات المطلوبة وجودها بصورة مستدامة.

تنقسم الفيوزات في مُتحكمات الـ AVR إلى 3 فيوزات رئيسية و- هي:
Low Fuse Byte : يتكون من 8 بิตات - كل 1 بت تتحكم في خاصية معينة في الـ AVR
HIGH Fuse Byte: أيضاً 8 بت - كل 1 بت تتحكم في مجموعة خصائص
Extended Fuse Byte: في معظم المُتحكمات تكون أقل من 5 بิตات

- جميع البتات في الفيوزات الثلاثة السابقة لها وضمان من البرمجة **programmed = 0** الفيوز تم برمجته وتفعيل الخاصية التي يتتحكم بها
- unprogrammed = 1** الفيوز لم يُفعل وتم إلغاء الخاصية التي يتتحكم بها

يتم تفعيل أحد الأوضاع بكتابة 0 أو 1 في هذا الفيوز، مع ملاحظة أن صفر لا تعني unprogrammed بل العكس - حيث يمكن أن الفيوز يتم تفعيله programmed عندما تضع بداخله القيمة 0 ويتم إلغاؤه عندما تضع بداخله القيمة 1.

تحذير: بعض الفيوزات إذا تم برمجتها بصورة خاطئة قد تتسبب في تعطيل المُتحكم أو إلغاء إمكانية برمجته مرة أخرى إلا باستخدام مبرمجات خاصة High voltage programmers، لذا كن على حذر عند اختيار قيم الفيوزات وتتأكد أكثر من مرة من جميع الإعدادات قبل برمجتها.

يمكنك التعرف على تفاصيل الفيوزات وكيفية تفعيلها وذلك عبر فتح الـ Datasheet الخاصة بمُتحكمات الـ AVR والبحث عن جداول Fuse LOW Byte , Fuse High Byte في صفحة 260 في حالة المُتحكم (ATmega16).



Fuse High Byte

Table 105. Fuse High Byte

Fuse High Byte	Bit No.	Description	Default Value
OCDEN ⁽⁴⁾	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN ⁽⁵⁾	6	Enable JTAG	0 (programmed, JTAG enabled)
SPIEN ⁽¹⁾	5	Enable SPI Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
CKOPT ⁽²⁾	4	Oscillator options	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	2	Select Boot Size (see Table 100 for details)	0 (programmed) ⁽³⁾
BOOTSZ0	1	Select Boot Size (see Table 100 for details)	0 (programmed) ⁽³⁾
BOOTRST	0	Select reset vector	1 (unprogrammed)

يوضح الجدول قيمة كل بت في جدول الفيوزات الـ High Byte مع القيمة الافتراضية لكل فيوز (هذه هي القيم التي تأتي مبرمجة مسبقاً من المصنع)، وكما نرى بعض الفيوزات تأتي مفعلاً بينما بعض يكون غير مفعلاً بصورة افتراضية.

قبل أن نبدأ مع الفيوزات سنحتاج أن نتعرف على مفهوم الـ Bootload

البوت-لودر هو برنامج (اختياري) صغير يتم تشغيله قبل البرنامج الرئيسي للقيام ببعض الوظائف، فمثلاً أشهر بووت-لودر هو Arduino bootloader هو السر الذي جعل تكافة لوحات آردوينو منخفضة جداً. يقوم هذا البوت-لودر بوظيفة بسيطة وهي فتح منفذ السيريرال UART ونقل البيانات (أن وجدت) إلى ذاكرة المتحكم flash memory هذا يعني أنه يقوم ببرمجة المتحكم عبر السيريرال بصورة ذاتية self programming دون الحاجة لاستخدام أي جهاز programmer وبالتالي يوفر كثيراً في تكافة صناعة اللوحات التطويرية ويعطي المتحكم إمكانية البرمجة الذاتية عبر منفذ السيريرال بدون تكلفة ذكر. فمثلاً لوحدة Arduino Nano تتبع من المواقع الصينية بـ 3 دولار فقط مما يجعلها أرخص لوحدة تطوير في العالم وتشمل المتحكم الدقيق USB to ttl converter atmega328 مع

والآن نعود مرة أخرى للفيوزات:

BOOTRST عند تفعيل هذا الفيوز يبدأ المتحكم الدقيق بتشغيل الـ bootloader المسجل في الذاكرة أولًا وبعد الانتهاء من تنفيذه يتم الانتقال إلى الـ main function، في أغلب الحالات لا يتم استخدام هذا الفيوز مالم تستخدم bootloader، القيمة الافتراضية = 1 مما يعني أن المتحكم سيشغل البرنامج الرئيسي وسيتجاهل البوت-لودر.

(0,1)BOOTSZ يتحكم هذا الفيوز في حجم ومكان تسجيل البوت-لودر في الذاكرة ولديهم جدول كامل من الإعدادات موجودة في صفحة 257 (مجدداً إذا لم تكن تتوافق استخدام بووت-لودر فلا تقم بتعديل إعدادات هذا الفيوز).

EESAVE يتحكم الفيوز في ما سيحدث لذاكرة الـ EEPROM عند برمجة المتحكم، عندما يتم وضع قيمته = 1 سيقوم المتحكم بمسح الـ EEPROM في كل مرة يتم رفع ملف هيكس جديد وعندما تكون قيمته = 0 فإن المتحكم لن يقوم بمسح محتوى الـ EEPROM أبداً مهما كان عدد مرات البرمجة. القيمة الافتراضية = 1



CKOPT يتحكم في طاقة المذبذب **clock oscillator** والمكثفات الداخلية (سيتم شرحه بالتفصيل في الجزء الخاص بالمذبذبات).

SPIEN تفعيل خاصية البرمجة عبر منفذ **SPI**، هذا الفيوز فائق الأهمية حيث يتحكم في تفعيل برمجة الذاكرة. وعندما يتم وضع قيمته = 1 يتم إلغاء إمكانية برمجة المتحكم مرة أخرى وعندها لن تستطيع أن ترتفع عليه أي برمج جديدة، القيمة الافتراضية = 0 وتعني أن البرمجة عبر **spi** تعمل. بعض الشركات تستخدم هذا الفيوز لمنع أي شخص من إعادة برمجة المتحكم الدقيق مرة أخرى.

لا تقم بـ**تبديل الفيوز SPIEN** أبداً طالماً أنك تتدرب على برمجة المعالج وإلا لن تتمكن من استخدام المتحكم مرة أخرى

JTAG, OCDEN كلا الفيوزان يستخدمان للتحكم في تفعيل خاصية التتحقق عبر بروتوكول **Jtag** العالمي.

ما هو **Jtag**

Jtag بروتوكول عالمي **Universal protocol** يتوفر في معظم المتحكمات المختلفة في العالم سواء كانت **PIC, AVR, ARM** أو حتى المصفوفات المنطقية **FPGA** ويستخدم في العديد من الأشياء المفيدة أهمها عملية التتحقق وقراءة محتويات المتحكم الداخلية.

فمثلاً قد تقوم بكتابة برنامج يقرأ قيمة مسجل ما ويتخذ قرار معين بناء على هذه القيمة لكن عند تنفيذ البرنامج نجد أن البرنامج يتصرف بصورة خاطئة، هذا التصرف الخاطئ يسمى **bug** – يعمل **Jtag debugger** على الوصول إلى محتويات المعالج وإيقافه في لحظة ما وقراءة جميع المحتويات مثل قيمة المتغيرات في الذاكرة أو محتوى المسجلات أو الأمر إلى سيد تنفيذه في **Jtag** **Program counter** .. الخ.

كما يمكن استخدام **Jtag** لتشغيل المتحكم بخاصية **single instruction execution** والتي تعني تنفيذ كل أمر بصورة يدوية، يعني أن المتحكم لن يقوم بتنفيذ الأوامر البرمجية بصورة متتالية ولكن سينفذ كل أمر على حدى عندما تأمره بذلك.

بهذه المميزات يساعدك **Jtag debugger** على تتبع الأخطاء واصلاحها والتتأكد من أن المتحكم يعالج البيانات بصورة صحيحة. كما أن بروتوكول **Jtag** يستخدم أيضاً في برمجة المتحكمات المختلفة وحتى **FPGA**. يمكنك قراءة المزيد عن هذا البروتوكول بصورة مفصلة من الرابط التالي:

en.wikipedia.org/wiki/Joint_Test_Action_Group

www.ATmel.com/webdoc/atmelice/atmelice.using_ocd_physical_jtag.html

لاحظ أن **ATmega16** يأتي مفعلاً بخيار تشغيل **Jtag** وهذا الخيار يلغى إمكانية استخدام معظم أطراف البورت **PORTC** كمنفذ **IN/OUT** لذا، إذا كنت تري استخدام هذا البورت بالكامل يجب عليك أن تلغى تفعيل **Jtag** بوضع قيمة **JTAGEN =1**



LOW Byte Fuse

Table 106. Fuse Low Byte

Fuse Low Byte	Bit No.	Description	Default Value
BODLEVEL	7	Brown-out Detector trigger level	1 (unprogrammed)
BODEN	6	Brown-out Detector enable	1 (unprogrammed, BOD disabled)
SUT1	5	Select start-up time	1 (unprogrammed) ⁽¹⁾
SUT0	4	Select start-up time	0 (programmed) ⁽¹⁾
CKSEL3	3	Select Clock source	0 (programmed) ⁽²⁾
CKSEL2	2	Select Clock source	0 (programmed) ⁽²⁾
CKSEL1	1	Select Clock source	0 (programmed) ⁽²⁾
CKSEL0	0	Select Clock source	1 (unprogrammed) ⁽²⁾

مجموعة الفيوzات الـ **0,1,2,3)CKSEL** (0,1,2,3) س يتم شرحها بالتفصيل في جزء المذبذبات

قبل شرح الفيوzات BOD يجب أن نتعرّف على مفهوم جديد أيضاً وهو الـ Brown Out Detection أو ما يعرف اختصاراً بـ BOD.

تعمل هذه الخاصية على إيقاف تشغيل المُتحكم الدقيق عندما يقل فرق الجهد المطبق عليه عن حد معين. وهذا سيدفعنا للتساؤل – **لماذا قد نحتاج أن نوقف تشغيل المُتحكم عندما يقل فرق الجهد؟؟**

العديد من المُتحكمات الدقيقة (ليس الـ AVR فقط) تتطلب فرق جهد معين ليعمل عند سرعة محددة بصورة مستقرة، فمثلاً المُتحكم ATmega16 يمكنه العمل بسرعة 1 ميجا بفرق جهد 1.8 فولت ولكن إذا تم تشغيله بسرعة 16 ميجا هرتز يجب أن يكون فرق الجهد الذي يعمل به = خمسة فولت على الأقل وحتى 5.5 فولت على الأكثر. وإذا انخفض الجهد عن الخامس فولت سيحدث اضطراب للمذبذب ولن يقوم بتوليد نبضات زمنية صحيحة.

بالنالي فإن أي أمر delay أو أي أمر معتمد على عامل زمني س يتم تنفيذه بصورة خاطئة. أيضاً انخفاض الجهد عند تشغيل المُتحكم على سرعات عالية قد يصبحه أخطاء في قراءة الذاكرة والـ EEPROM وقد يتسبب في أخطاء أخرى عشوائية. لذا يتم استخدام الـ BOD لإيقاف تشغيل المُتحكم إذا كان فارق الجهد المطبق عليه أقل من المطلوب ويتم إعادة تشغيله تلقائياً عندما يعود فارق الجهد للمستوى المناسب ليعمل المُتحكم بكفاءة.

الحقيقة أنه لا داعي لتشغيل الـ BOD إلا إذا كنت س تشغيل المُتحكم بسرعة أكبر من 4 ميجا هرتز لأن معظم مُتحكمات الـ AVR يمكنها العمل بصورة طبيعية حتى فرق جهد 1.8 فولت على سرعات من 1 إلى 4 ميجا.

BODEN يقوم هذا الفيوز بتفعيل خاصية الـ BOD ويأتي بقيمة 1 (غير مفعول) بصورة افتراضية **BODLEVEL** يتحكم الفيوز في فارق الجهد الذي س يتم إيقاف المُتحكم عن التشغيل عندما يصل إليه، فإذا كانت قيمة الفيوز = 1 هذا يعني أن المُتحكم س يتم إيقاف تشغيله عند فرق جهد أقل من 2.7 فولت أما إذا كانت قيمة الفيوز = 0 هذا يعني أن المُتحكم س يتوقف عند فرق جهد أقل من 4 فولت (مع ملاحظة أن هذه الإعدادات تصبح فعالة إذا كانت قيمة الـ BODEN مفعولة أي أنها تساوي صفر).

SUT0,1 تتحكم هذه الفيوزات في الزمن الذي سيستغرقه المُتحكم من بداية توصيل الكهرباء حتى البدء في تنفيذ البرنامج المُسجل بداخله أو الوقت المستغرق لعمل Reset. يمكنك ضبط هذا الفيوز بمجموعة من القيم المختلفة (الجدول يوضح زمن تأخير التشغيل عند استخدام المذبذب الداخلي مع ملاحظة أن هذه الإعدادات تختلف عند استخدام كريستالة خارجية أو مذبذب من نوع آخر).

**Table 10.** Start-up Times for the Internal Calibrated RC Oscillator Clock Selection

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset ($V_{CC} = 5.0V$)	Recommended Usage
00	6 CK	—	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10 ⁽¹⁾	6 CK	65 ms	Slowly rising power
11	Reserved		

كما هو موضح في الجدول، نجد أن تغيير الإعدادات سيجعل المُتحكم يتأخّر إما 0 أو 4 أو 65 ملي ثانية عند بداية التشغيل (يعني أنه سيستغرق 64 ملي ثانية من بدء توصيل الكهرباء إلى أن يبدأ تنفيذ البرنامج).

لماذا نحتاج أن نجعل المُتحكم يتأخّر فترة ما قبل أن يبدأ تنفيذ البرنامج؟

هناك سببين لهذا الأمر.

- الأول: بعض الدوائر والعناصر الإلكترونية التي قد تتصل بالمُتحكم الدقيق تحتاج القليل من الوقت بعد توصيل الكهرباء حتى تكون جاهزة للعمل بصورة مستقرة لذا يستحسن أن يتأخّر المُتحكم الدقيق قليلاً قبل أن يبدأ العمل أو الاتصال مع هذه المكونات.
- الثاني: معظم دوائر المذبذبات Oscillator التي تولد الـ CPU Clock تحتاج لوقت كثير نسبياً حتى تستقر النبضات المترددة منها مثل دوائر الـ RC oscillator (ceramic oscillator) لذا يجب على المُتحكم أن يتأخّر قليلاً حتى تستقر نبذبات الساعة الداخلية.

يستحسن أن تجعل التأخير الزمني = 65 ملي ثانية لضمان أفضل أداء للمُتحكم وهذا يعني أن تجعل قيمة الـ SUT0 = 1 و SUT1 = 1 كلاهما



LockBits 6.2

مثل الفيوزات تحكم الـ lockbits في مجموعة من الخصائص الثابتة التي لا تتغير إذا تم برمجتها، هذه الخصائص هي "إمكانية الوصول وحماية الذاكرة". حيث يمكنك استخدام الـ lockbits في حماية البيانات على ذاكرة المتحكم الرئيسية من النسخ أو القراءة كما يمكنك حماية بعض أجزاء الذاكرة من الكتابة عليها ومنع أي برنامج أو شخص من الوصول إلى محتواها.

أيضاً تستخدم الـ lockbits في تخصيص جزء ثابت من الذاكرة لا يقبل التعديل بعد برمجته مثل ما يحدث مع الـ bootloader (أشهر مثال للbootloader هو arduino bootloader) وهو البرنامج المسؤول عن استقبال ملف الـ hex من السيريرال بدل من الـ SPI.

كما نرى في الجدول التالي اعدادات الـ lockbit التي يمكنها أن تغلق إما القراءة والكتابة أو القراءة فقط من جميع أنواع الذاكرة Flash & EEPROM وذلك عبر التحكم في قيم الـ LB1 و LB2.

Table 104. Lock Bit Protection Modes

Memory Lock Bits ⁽²⁾			Protection Type
LB Mode	LB2	LB1	
1	1	1	No memory lock features enabled.
2	1	0	Further programming of the Flash and EEPROM is disabled in Parallel and SPI/JTAG Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode. ⁽¹⁾
3	0	0	Further programming and verification of the Flash and EEPROM is disabled in Parallel and SPI/JTAG Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode. ⁽¹⁾

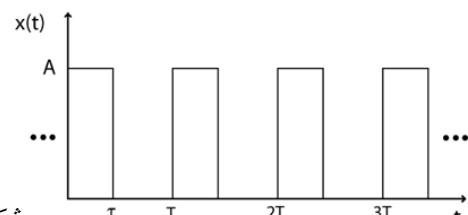
- عندما تكون 1 = LB1 و 1 = LB2 لا يتم تفعيل الـ lockbit ويسمح بالقراءة والكتابة من وإلى الذاكرة.
- عندما تكون 0 = LB1 و 1 = LB2 يتم تفعيل الحماية ويمنع أي محاولة (كتابة) بيانات على كل من الـ Flash و الـ EEPROM.
- عندما تكون 0 = LB1 و 0 = LB2 يتم تفعيل الحماية ويمنع أي محاولة (كتابة أو قراءة) البيانات على كل من الـ EEPROM و الـ Flash.

Clock Source 6.3

عندما نشتري أجهزة الحاسوب الخاصة نحاول دائماً شراء الجهاز الأسرع وهذا نجد الشركات المصنعة دائماً تتيهانى بأن منتجاتها هي الأفضل لأنها تعمل بأعلى سرعة معالجة (تقاس بالجيجاهرتز - ملليار هرتز). ومثل أجهزتنا الشخصية نجد أن سرعة المتحكمات الدقيقة هي أيضاً تحدد عبر التردد الذي تعمل عليه أو كما تسمى (مصدر الساعة clock source).

حيث يتم تنفيذ الأوامر بسرعة = $1 \div (\text{التردد})$ الذي يعمل به المتحكم، على سبيل المثال. جميع شرائح الـ AVR تقوم بتنفيذ أمر واحد كل 1 نبضة من الـ clock source.

شكل نبضات الساعة عبر الزمن





- إذا كان التردد = 1 ميجا هرتز (السرعة الافتراضية لمعظم مُتحكمات الـ AVR) هذا يعني أن المُتحكم يمكن تنفيذ 1 مليون أمر بلغة الأسمبلي في الثانية الواحدة ويستغرق الأمر 1 ميكروثانية
- إذا كان التردد = 1 ميجا هرتز هذا يعني أن المُتحكم يمكنه تنفيذ 1 مليون أمر بلغة الأسمبلي في الثانية الواحدة ويستغرق الأمر 1 ميكروثانية
- إذا كان التردد = 2 ميجا هرتز هذا يعني أن المُتحكم يمكنه تنفيذ 2 مليون أمر بلغة الأسمبلي في الثانية الواحدة ويستغرق الأمر نصف ميكروثانية أو (500 نانو ثانية).
- إذا كان التردد = 4 ميجا هرتز هذا يعني أن المُتحكم يمكنه تنفيذ 4 مليون أمر بلغة الأسمبلي في الثانية الواحدة ويستغرق الأمر 250 نانو ثانية **nano second**
- إذا كان التردد = 8 ميجا هرتز هذا يعني أن المُتحكم يمكنه تنفيذ 8 مليون أمر بلغة الأسمبلي في الثانية الواحدة ويستغرق الأمر 125 نانو ثانية

النano ثانية (nS) = جزء من مليار جزء من الثانية

الميكرو ثانية (uS) = جزء من مليون جزء من الثانية.

تدعم مُتحكمات الـ AVR العديد من تقنيات توليد النبضات اللازمة لتشغيل المُتحكم أو كما تسمى **Clock Source** تختلف هذه التقنيات في مدى دقتها ونسبة الخطأ والتكلفة المالية، في هذه الجزء سنستعرض 4 طرق مختلفة مع إيضاح مميزات وعيوب كل طريقة.

Internal Calibrated RC oscillator

كلمة **RC oscillator** تعني "المذبذب الداخلي المصنوع بمقاومة ومكثف" وتعتبر هذه الطريقة هي أرخص أسلوب للحصول على الـ **Clock source** حيث تحتوي شرائح الـ AVR ومنها **ATmega16** على مذبذب **RC** داخلي يمكنه العمل بسرعات من 1 إلى 8 ميجا هرتز (يتم ضبطها بالفيوزرات كما سنرى لاحقاً). ولا تتطلب هذه الطريقة توصيل أي مكونات إضافية وبالتالي يتم تصغير حجم المشروع وتقليل التكلفة

العيوب: نسبة الخطأ في التردد والتوقيت 3% - هذه النسبة قد لا تؤثر في معظم التطبيقات، لكنها خطيرة جداً في التطبيقات التي تحتاج توقيت دقيق أو تستخدم دوال **delay_ms** على مفترات بعيدة. على سبيل المثال: إذا كان مشروعك عبارة عن تشغيل **load** كل ثانية واحدة فلن تشعر بهذا الخطأ لأن التوقيت الحقيقي = 1 ثانية (1+) - 1 ثانية (1-) = نسبة الخطأ = 0.03 * 1 = 0.03 و هو ما يساوي 1.03 ثانية أو 1 ثانية طرح 0.03 = 0.97 ثانية.

لكن تخيل أن مشروعك يقوم بتشغيل **load** كل 1 ساعة = 3600 ثانية، عندها ستجد أن الوقت الحقيقي يصبح $1 + 3600 * 0.03$ (3600) وهو ما يساوي 3708 ثانية (هذا يعني أنه هناك تأخير أو تقديم) 108 ثانية إضافية والذي يعتبر رقم كارثي و بعيد تماماً عن التوقيت المطلوب. متى نستخدمها؟ إذا كان مشروعك لن يتواجد به أي تأخير زمني يزيد عن بضعة ثواني فقط (أقل من 10 ثواني) وتريد أن تصغر تكلفة المشروع.

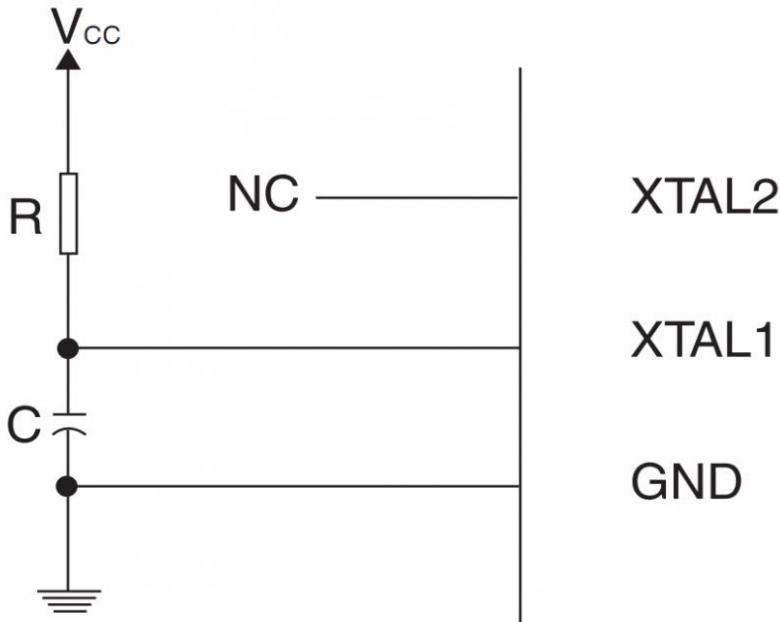
External RC Circuit

هذه الطريقة مماثلة للسابقة ولكن الاختلاف الوحيد أن دائرة الـ **RC** تكون موجودة خارج شريحة الـ AVR وتتوفر ترددات أكبر من 8 ميجا حيث يمكنك ضبط التردد عبر التحكم في قيمة المقاومة والمكثف عبر القانون التالي:

حيث تمثل **R** قيمة المقاومة بالأوم، و **C** قيمة المكثف بالفاراد ويتم توصيل الدائرة على الطرف XTAL1 بالطريقة التالية



(NC يعني غير متصل بشيء)



المميزات: يمكنك الحصول على ترددات أكبر من الداخلي internal oscillator بتكلفة قليلة كما يمكنك تغيير التردد في أي وقت عبر وضع مقاومة متغيرة بدلاً من المقاومة الثابتة.

العيوب: مثل النوع السابق، غالباً لا يمكنك تحديد قيمة المكثف ولا المقاومة بدقة كبيرة حيث تحتوي هذه العناصر على نسبة خطأ 5% وبالتالي التردد الحقيقي الناتج عنها يكون مضاد إليه نسبة خطأ 5%.

في حالة أنك تريد استخدام هذه الطريقة يجب الانتباه إلى قيمة المكثف حيث يجب أن تكون 22 بيكوفاراد أو أكثر ويجب أن يكون من النوع السيراميك (منعدم القطبية) ceramic capacitor.

معلومة إضافية: بعض مُتحكمات AVR تحتوي على مكثف داخلي بقيمة 36 بيكوفاراد وبالتالي لن تحتاج سوى أن توصل مقاومة فحسب.

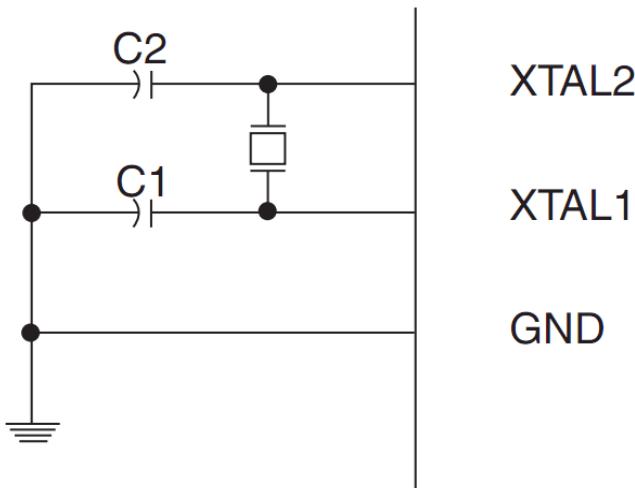


External Crystal “Quartz” Oscillator

تعتبر هذه الطريقة هي الأكثر شيوعاً والمفضلة لدى جميع الشركات ومصممي الأنظمة المدمجة، حيث يتم استخدام الكريستالة ذات الطرفين لتوليد التردد المطلوب بدقة عالية جداً وبأقل نسبة خطأ ممكنته حيث تبلغ نسبة الخطأ في الكريستالة حوالي 10 هرتز من أصل 1 مليون وهو ما يساوي 0.00001 واحد من كل مئة ألف وهذا يعني أنها أكثر بدقة بحوالي 1000 مرة من الـ RC oscillator.

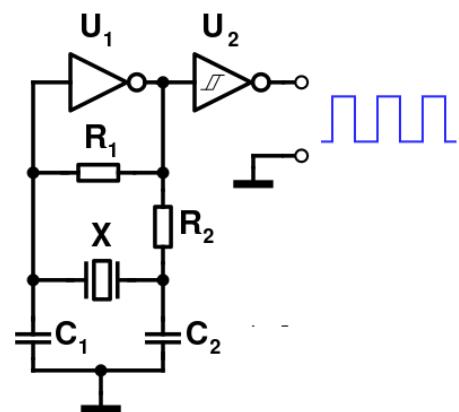
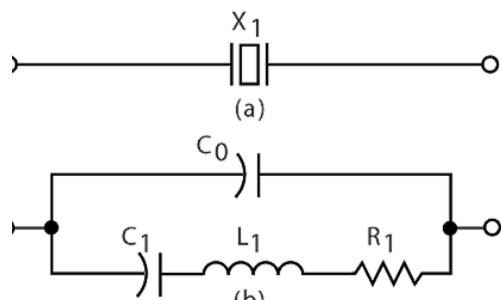
المميزات: الدقة العالية جداً وثبات التردد مهما تغيرت درجات الحرارة وبالتالي فهي توفر أداء ممتاز طوال فترة التشغيل
العيوب: التكلفة حيث تحتاج هذه الكريستالات إلى مكثفات إضافية عدد 2 مكثف بسعة 22 بيكوفاراد مما يزيد التكلفة لتصل إلى 1 دولار تقريباً (تكلفة الكريستالة بمفردها = نصف دولار).
أشهر هذه الكريستالات هي 16 ميجا والـ 12 ميجا والـ 8 ميجا والـ 24 ميجا، في هذا الكتاب سنستخدم الـ 16 ميجا في بعض التجارب (وهو أقصى تردد يمكن لشريحة الـ ATmega16 أن تعمل به مع العلم أن المتحكم ATTiny يمكنه العمل بسرعة تصل إلى 20 ميجاهرتز).

يتم توصيل الكريستالة والمكثفات بالصورة التالية:



معلومة إضافية: الكريستالة هي عنصر إلكتروني مصنوع من مادة بلورية اسمها "الكوارتز Quartz" وعندما يتم توصيلها بالكهرباء فإنها تعمل كأنها دائرة رنين مكونة من مكثف وملف ومقاومة R-L-C circuit تتنتج Sin wave ولكن عندما تتصل بدائرة pierce oscillator داخل المتحكم الدقيق يتم تحويل الذبذبات المتولدة على هيئة pulse إلى sin wave باستخدام عاكس inverter يمكنه المزيد عن الـ pierce oscillator من الرابط التالي:
http://www.abracon.com/Support/facn_abracon_jul2011.pdf

ويمكنك قراءة صفحة الكريستالات على ويكيبيديا لتعرف أكثر عليها.
en.wikipedia.org/wiki/Crystal_oscillator



تركيب دائرة المذبذب (الكريستالة + المكثفات + العاكس)



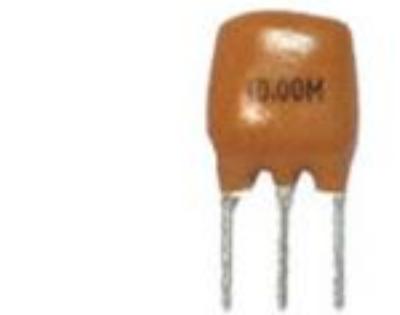
External Resonators

هذا النوع من المذبذبات يشبه لحد كبير الكريستالة وفي الحقيقة هو يوازي "كريستالة + المكثفات الإضافية" في قطعة إلكترونية واحدة ويتوفّر في الأسواق بترددات مختلفة بدأ من 1 ميجاهرتز إلى 24 ميجاهرتز

السعر
(أرخص من
كما أنه يحتوي
المطلوبة
الصغرى
الترددات



نسبة خطأ
بالمئة)، وهي
أفضل بكثير
internal RC
ولكنها لا



CERAMIC RESONATOR

المميزات:
المنخفض
الكريستالة
على المكثفات
بداخله وحجمه
توافر معظم
المطلوبة

العيوب: يتملك
نسبة 0.5% (نصف
الـ oscillator
تعتبر نسبة
من الـ
قارن بدقة الكريستالة الكوارتز.

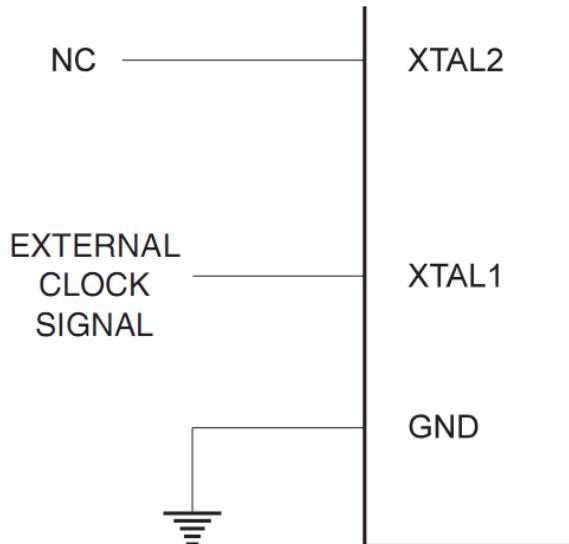
External Pure Pulse (TTL) Oscillator

المذبذبات الأكثر دقة على الإطلاق حيث تبلغ نسبة الخطأ 5 هرتز لكل 1 مليون هرتز وهو ما يساوي 0.000005 و الذي يعني أنها تمتلك دقة توافي 20 ضعف دقة الكريستالات الكوارتز.

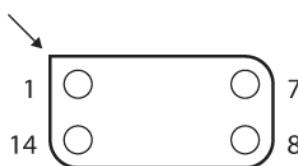
تتوفر هذه المذبذبات بشكال عديدة أغلبها تكون مربعة أو مستطيله الشكل و تمتلك 4 أطراف VCC - CLK - GND - XTAL1 في شريحة المتحكم AVR و توصيل الطرف CLK بنفس مصدر الجهد الخاص بالـ AVR (سواء كان 5 فولت أو 3.3 فولت).



العيوب: السعر المرتفع، حيث يبلغ سعر هذا النوع من المذبذبات نحو 3 دولار أو أكثر (يزداد السعر بزيادة التردد المطلوب وفي بعض الأحيان يكون سعرها أكبر من سعر المتحكم الدقيق نفسه. الصور التالية توضح توصيل الـ AVR مع المذبذب TTL oscillator



الصور التالية توضح أشكال وأحجام مختلفة للمذبذب



Pin	Function
1	NC
7	GND
8	Output
14	+5VDC

6.4 قيم الفيوزات لضبط السرعة

القيم التالية هي المستخدمة في التحكم بسرعة شريحة ATmega16 - جميع القيم تم أخذها من ملف DataSheet الرسمي للمتحكم (والمرفق مع الكتاب) بداية من الصفحة رقم 24 مع ملاحظة أن هذه الفيوزات تعمل على كل من لأنهم من نفس القمة من المُتحكمات (وقد تختلف هذه الفيوزات في فئات أخرى من شرائح ATmega16/ ATmega32 CKSEL). - جميع القيم مخصصة للفيوزات Clock Select أو كما تعرف اختصاراً بي AVR



فيوزات ضبط السرعة مع المذبذب الداخلي Internal RC

قم بضبط الفيوزات بالقيم التالية لاختيار السرعة المطلوبة، مع ملاحظة أن القيمة هي „، CKSEL3، CKSEL2، CKSEL1، CKSEL0“ بنفس الترتيب حيث يعبر كل 1 بت عن قيمة الـ CKSEL الموازية له.

(3,2,1,0)CKSEL	Clock Frequency
0001	1 Mhz
0010	2 Mhz
0011	4 Mhz
0100	8 Mhz

فيوزات ضبط السرعة مع دائرة External RC

(3,2,1,0)CKSEL	Clock Frequency
0101	التردد أقل من 0.9 ميجا هرتز
0110	التردد المتوقع أكبر من 0.9 وأقل من 3 ميجا هرتز
0111	التردد المتوقع أكبر من 3 وأقل من 8 ميجا هرتز
1000	التردد المتوقع أكبر من 8 وأقل من 12 ميجا هرتز

فيوزات ضبط السرعة مع External Crystal or Ceramic

هذه الإعدادات يمكن استخدامها مع كل من الـ Crystal oscillator والـ ceramic resonator مع ملاحظة أنه في حالة استخدام الكريستال يجب أن يتم وضع مكثفات إضافية معها كما ذكرنا سابقاً ولا يتم وضع هذه المكثفات مع الـ ceramic resonator.

أيضاً لاحظ أن الإعدادات يتم وضعها للفيوزات CKSEL من 1 إلى 3 فقط ولا يتم برمجة الـ CKSEL0 معهم في هذا الوضع. ويتم برمجة الفيوز CKOPT في حالة استخدام كريستال أكبر من 8 ميجا هرتز.

CKSEL(3,2,1) "0 is not used"	Clock Frequency	سعة المكثفات المقترحة
101 CKOPT = 1	ceramic resonator هذا الوضع يستخدم مع وفي حالة أن التردد المطلوب ما بين 0.4 إلى 0.9 ميجا هرتز	لا يتم استخدام مكثفات
110	التردد المتوقع أكبر من 0.9 وأقل من 3 ميجا هرتز	السعة المقترحة بين 12 → 22 picofarad



CKOPT = 1		
111	التردد المتوقع أكبر من 3 وأقل من 8 ميجاهرتز	السعة المقترنة بين 12 → 22 picofarad
CKOPT = 1		
111	التردد المتوقع أكبر من 8 ميجا وحتى 16 ميجاهرتز	السعة المقترنة بين 12 → 22 picofarad
CKOPT = 0		



ما هي أهمية الفيوز CKOPT؟

هذا الفيوز يتحكم في بعض الأمور المهمة، منها تشغيل المذبذب بالطاقة القصوى أو الطاقة المنخفضة (وضع توفير الطاقة عندما يكون $CKOPT = 1$ "unprogrammed" fuse) هذا الوضع يساهم في تخفيف الطاقة التي يستهلكها المتحكم الدقيق لكنه لا يصلح لتشغيل الكريستالات الأكبر من 8 ميجاهرتز لأنها كلما زاد تردد الكريستالة كانت النبضات الناتجة منها أضعف من ناحية فرق الجهد وبالتالي قد لا تصلح لتشغيل المعالج.

للتغلب على هذه المشكلة يتم تفعيل الـ CKOPT والذي سيقوم بتشغيل المذبذب بالطاقة القصوى أو كما يسمى- Full rail-to-rail swing مما يجعل المعالج يحصل على أفضل نبضات ممكنه تكفي لتشغيله في السرعات العالية وتكتفى أيضاً بإخراج نبضات دقيقة لتشغيل المكونات الخارجية.

من المفيد جداً تفعيل هذا الفيوز في الدوائر التي تتعرض إلى noise أو ستوضع في مكان معرض لإشعاع كهرومغناطيسي كبير نسبياً حيث يساعد وضع Full rail-to-rail على تحسين أداء المتحكم في البيانات ذات الـ noise الكبيرة.

الوظيفة الثانية له هي تفعيل المكثف الداخلي "سعة 36 بيكوفاراد" لتشغيل دائرة الـ external RC circuit بالمقاومة فقط دون مكثف إضافي.

فيوزات ضبط السرعة مع المذبذب الخارجي Pulse Oscillator

عند استخدام أي مصدر خارجي لنبضات الساعة pulses مثل الـ TTL oscillator أو حتى شريحة IC 555 يتم وضع كل قيم CKSEL(3,2,1,0) = 0000



ملخص إعدادات الفيوزات للتحكم بالسرعة

- إذا أردت تشغيل المتحكم الدقيق بأقل استهلاك ممكن للطاقة قم بتعطيل التردد على 1 ميجاهرتز فقط عبر وضع .CKSEL(3,2,1,0) = 0001
- إذا أردت تشغيل المتحكم بأقصى سرعة (8 ميجاهرتز) دون استخدام أي مكونات إضافية قم بوضع القيمة .CKSEL(3,2,1,0) = 0100
- إذا كنت تتوسيع أن تستخدم أي كريستال خارجية فالأفضل أن تشغيل الـ Full rail-to-rail CKOPT بوضع القيمة 0 بداخله و اختيار 111 = CKSEL(3,2,1) – هذا الوضع سيعمل بصورة ممتازة مع جميع الكريستالات.

ملاحظات هامة بخصوص تعديل السرعة

يجب الانتباه عند تغيير سرعة المتحكم الدقيق حيث أن سرعة رفع البرنامج (ملف الهايكس) يجب أن تكون أقل من 1/8 من تردد الـ **Clock** المستخدم.

على سبيل المثال إذا كانت سرعة المتحكم = 1 ميجاهرتز إذا يجب أن تكون سرعة رفع البرنامج على المتحكم أقل من 128 كيلوهرتز وإلا قد تجد خطأ من برنامج AVRdude مفاده أن البرنامج لا يستطيع أن يتواصل مع المتحكم الدقيق.

على أي حال إذا قرأت هذا الخطأ فكل ما عليك فعله هو تقليل سرعة الرفع عن طريق توصيل طرفي الـ jumper الموجودة بدائرة المبرمج USBasp (ستجد كلمة **slow** مكتوبة بجانبها) وإذا كنت تستخدم مبرمجة أخرى لا تحتوى هذا الوضع فيمكنك أن تختار سرعة الرفع من برنامج AVRdudess (اجعل قيمتها = 500 هرتز فقط).

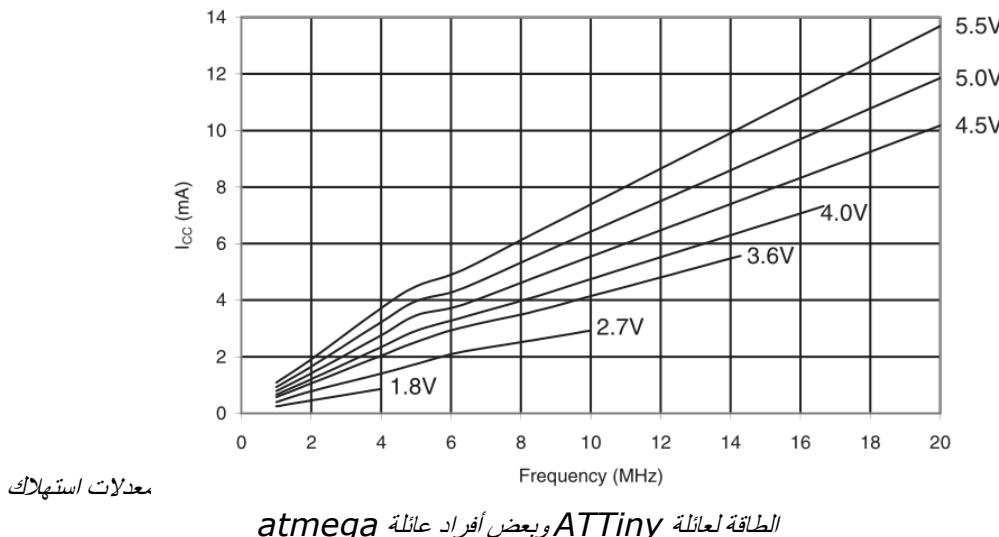
عند استخدام الدالة `delay_us(time)` يجب أن يعمل المذنب بكريستال خارجية بتردد 8 ميجاهرتز على الأقل حتى تعمل الدالة بتوقيت صحيح (إذا تم استخدام المذنب الداخلي أو كريستال بتردد أقل فيحدث خطأ في الدالة `delay_us` ولن يتم التأخير بالوقت المطلوب)



6.5 الطاقة وسرعة تشغيل المُتحكمات

في العديد من التطبيقات يجب أن يتم تصميم النظام المدمج بحيث يعمل على مصدر منخفض جداً للطاقة مثل بطارية أو لوحة شمسية صغيرة، من أجل هذه التطبيقات يتم ضبط المُتحكم للعمل على الترددات المنخفضة.

معظم المُتحكمات الدقيقة (وبالتحديد المعالجات الدقيقة الموجودة بداخلها) تستهلك طاقة أكثر كلما كانت تعمل بتردد أكبر وبالتالي نجد أن تخفيض التردد يقل بدرجة كبيرة جداً استهلاك التيار الكهربائي كما هو ملحوظ في الرسم البياني التالي:



كما نرى على المحور الرئيسي (استهلاك التيار بالملي أمبير) والمحور الأفقي التردد (بالميجاهرتز) وكل خط مرسم يمثل فرق الجهد الذي يعمل عنده المُتحكم الدقيق. من هذا الرسم نستنتج العديد من الأمور.

- عند فرق جهد 1.8 فولت يمكن للمُتحكم أن يعمل بتردد من 1 ميجا إلى 4 ميجا بحد أقصى وباستهلاك تيار من الثاني حتى 1.8 مللي أمبير فقط.

- عند فرق جهد 2.7 فولت يمكن للمُتحكم أن يعمل بسرعة تصل إلى 10 ميجاهرتز واستهلاك تيار حوالي 3 مللي أمبير.

- عند فرق جهد 3.6 فولت يمكن للمُتحكم أن يعمل بتردد يصل إلى 14.5 ميجاهرتز تقريباً وباستهلاك تيار يصل إلى 5.7 مللي أمبير.

- عند فرق جهد 4 فولت يمكن للمُتحكم أن يعمل حتى 17 ميجاهرتز باستهلاك تيار 7 مللي أمبير.

- عند فرق جهد من الخامس إلى 5.5 يمكن للمُتحكم أن يعمل بسرعة تصل إلى 20 ميجاهرتز.

ملحوظة: الحد الأقصى لتردد آلة ATMega16/ATMega32 هو 16 ميجاهرتز فقط، بينما المُتحكمات الأحدث منها مثل آلة ATTiny أو آلة ATMega328 يمكنها أن تصل إلى 20 ميجاهرتز



بعض مُتحكمات الـ AVR تدعم العمل بتردد 125 كليوهرتز أو أقل، هذا المعدل يجعل استهلاك التيار الكهربائي منخفض جداً لدرجة أنه يصل إلى 100 ميكرو أمبير (100 جزء من المليون من الأمبير وهو ما يساوي 0.1 مللي أمبير).

تختلف هذه المقاييس قليلاً بتغير درجة الحرارة التي يعمل عنها المعالج وتوجد رسومات بيانية مفصلة تشرح معدلات استهلاك الطاقة بالتفصيل عند درجات الحرارة المختلفة بدءاً من صفحة 299 في الـ Datasheet المرفقة (مع ملاحظة أن بعض معدلات استهلاك الطاقة قد تختلف قليلاً مع مُتحكمات الـ AVR الأخرى).

كيف تحسب عمر البطارية

عند استخدام البطاريات سيكون من المفيد جداً أن تحسب وقت التشغيل حتى تنفذ طاقة البطارية وقد يحدد هذا الوقت التردد المطلوب لتشغيل المُتحكم. وقبل أن نبدأ الحسابات علينا أن نتعرف على بعض الأمور.

استهلاك الطاقة المذكور مسبقاً هو للمُتحكم الدقيق نفسه وليس لأي حمل load متصل به ويمكن تخفيض هذا الاستهلاك قليلاً بإيقاف تشغيل الـ ADC.

استهلاك الطاقة الكلي $Total load$ = فرق الجهد * (استهلاك التيار للمُتحكم + الأحمال المتصلة به) + الطاقة الضائعة من الـ voltage regulator إن وجد.

$$\text{قانون Battery Working Time} = \frac{0.8 \times \text{Battery Capacity (mAH)}}{\text{Total Load Current (mA)}} \text{ البطارية}$$

جميع البطاريات يكون لها سعة تفاصيل بالمللي أمبير (أمسية) ساعة فمثلاً بطارية الهاتف المحمول نجد مكتوب عليها 3.7 فولت mAH 3000 (مللي أمبير ساعة) أو تكتب 1 A/hour (لأن كل 1000 مللي أمبير = 1 أمبير). وهذا يعني أنه في حالة تشغيل هذه البطارية على أحمال تستهلك 3000 مللي أمبير فإن البطارية ستظل تعمل 1 ساعة فقط.

مثال: إذا كان المُتحكم الدقيق يعمل بفرق جهد 3.7 فولت و بتردد 16 ميجاهرتز (يستهلك 7 مللي أمبير) ومتصل به دايوه ضوئي يستهلك 7 مللي أمبير كما أن منظم الجهد يستهلك تيار إضافي 6 مللي أمبير. احسب زمن التشغيل على بطارية سعتها 1000 مللي أمبير بافتراض أن جميع الأحمال تعمل بصورة مستمرة دون أن تتطفىء.

الحل
أولاً: نحسب استهلاك التيار الكلي = 7 مللي (للمُتحكم) + 7 مللي (للدايو) + 6 مللي لمنظم الجهد = 20 مللي أمبير إجمالي استهلاك طاقة.

ثانياً: بالتعويض بالقيم في القانون السابق نجد أن ساعات التشغيل = 40 ساعة

$$\text{Working Time} = 0.8 * (1000/20) = 40 \text{ hours}$$

معلومة إضافية: لماذا نضرب الرقم 0.8 في القانون السابق بالرغم من أنه يفترض أن نقسم سعة البطارية على الاستهلاك مباشرة؟ السبب هو توفر استهلاك ضائع من الطاقة يحدث من البطارية نفسها بسبب **المقاومة الداخلية Internal Resistor** كما أن الأسلام والتوصيلات في الدائرة الكهربائية أيضاً تساهم في ضياع بعض الطاقة خاصة إذا كان هناك أسلام من معدن الألومنيوم

6.6 كيف تبرمج الفيوزات

برمجة الفيوزات عملية سهلة للغاية عبر برنامج AVRdudeess كل ما عليك فعله هو اختيار نوع المُتحكم الدقيق وأداة البرمجة programmer



Programmer (-c)

USBtiny simple USB programmer, <http://www.ladyada.net/make/usbtinyisp/>

Port (-P)	Baud rate (-b)	Bit clock (-B)
<input type="button" value="▼"/>	<input type="text"/>	<input type="text"/>

Flash

<input checked="" type="radio"/> Write	<input type="radio"/> Read	<input type="radio"/> Verify
<input type="button" value="Go"/>	<input type="button" value="Format"/>	<input type="button" value="Auto (writing only)"/>

EEPROM

<input checked="" type="radio"/> Write	<input type="radio"/> Read	<input type="radio"/> Verify
<input type="button" value="Go"/>	<input type="button" value="Format"/>	<input type="button" value="Auto (writing only)"/>

Options

<input type="checkbox"/> Force (-F)	<input type="checkbox"/> Erase flash and EEPROM (-e)
<input type="checkbox"/> Disable verify (-V)	<input type="checkbox"/> Do not write (-n)
<input type="checkbox"/> Disable flash erase (-D)	Verbosity <input type="button" value="0"/>

Program! **Additional settings**

MCU (-p)

ATmega16

Flash: 16 KB
EEPROM: 512 B

Presets

Default

Fuses & lock bits

L	<input type="text"/>	<input type="button" value="Read"/>	<input type="button" value="Write"/>
H	<input type="text"/>	<input type="checkbox"/> Set fuses	<input type="button" value="Fuse settings"/>
E	<input type="text"/>	<input type="button" value="Read"/>	<input type="button" value="Write"/>
LB	<input type="text"/>	<input type="button" value="Read"/>	<input type="button" value="Write"/>
<input type="checkbox"/> Set lock			
Bit selector			

ثم الضغط على زر Bit selector من مربع لوحة اختيار قيم الفيوزات (لاحظ أن Fuses & lockbits لا يمتلك ATmega16)

Lock Bits

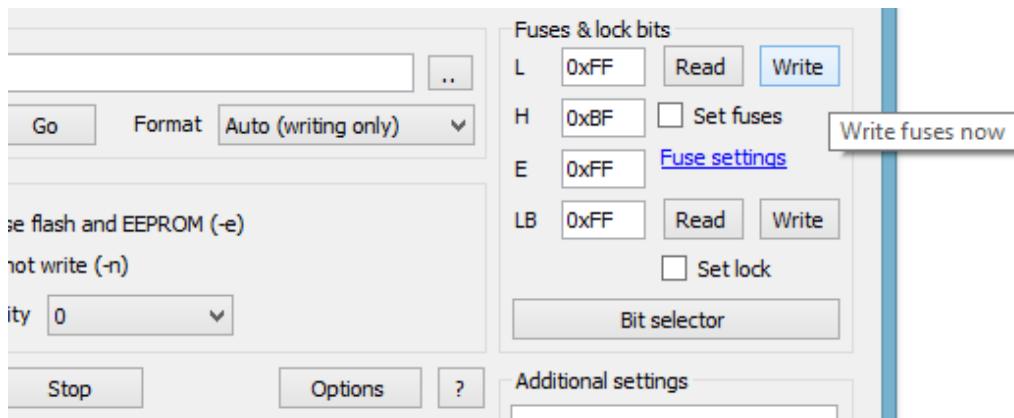
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> BLB12	<input type="checkbox"/> BLB11	<input type="checkbox"/> BLB02	<input type="checkbox"/> BLB01	<input type="checkbox"/> LB2	<input type="checkbox"/> LB1	LOCK BITS 0xFF
--------------------------	--------------------------	--------------------------------	--------------------------------	--------------------------------	--------------------------------	------------------------------	------------------------------	-------------------

Fuse bits

BODLEVEL <input type="checkbox"/> 1	BODEN <input type="checkbox"/> 1	SUT1 <input type="checkbox"/> 1	SUT0 <input type="checkbox"/> 1	CKSEL3 <input type="checkbox"/> 1	CKSEL2 <input type="checkbox"/> 1	CKSEL1 <input type="checkbox"/> 1	CKSEL0 <input type="checkbox"/> 1	LFUSE 0xFF
OCDEN <input type="checkbox"/> 1	JTAGEN <input type="checkbox"/> 0	<input type="checkbox"/> 1	CKOPT <input type="checkbox"/> 1	EESAVE <input type="checkbox"/> 1	BOOTSZ1 <input type="checkbox"/> 1	BOOTSZ0 <input type="checkbox"/> 1	BOOTRST <input type="checkbox"/> 1	HFUSE 0xBF
<input type="checkbox"/> 1	<input type="checkbox"/> 1	<input type="checkbox"/> 1	<input type="checkbox"/> 1	<input type="checkbox"/> 1	<input type="checkbox"/> 1	<input type="checkbox"/> 1	<input type="checkbox"/> 1	EFUSE 0xFF

OK

اختر تفعيل (أو إلغاء تفعيل) الفيوزات المرغوبة كما تريده ثم اضغط Ok لتجد أن قيمة الـ HIGH Low Byte Fuse والـ Byte Fuse أصبحت جاهزة للبرمجة كما في الصورة التالية



الآن يمكنك الضغط على زر Write الموجود في الأعلى (جانب الفيوزات) وسيقوم الـ programmer بحرق فيم الفيوزات المطلوبة.

يوفر برنامج AVRdudeess خيار Set fuses والذي يعني أن البرنامج سيقوم بإعادة كتابة الفيوزات في كل مرة يقوم فيها برفع ملف هيكس جديد على المتحكم الدقيق. من الأفضل عدم استخدام هذا الخيار والاكتفاء ببرمجة الفيوزات عبر زر Write فحسب.

أيضاً الزر Read بجانب Write والذي عند الضغط عليه سيقوم الـ Programmer بقراءة قيمة الفيوزات على الشريحة المتصلة به.

أيضاً ستحد كلمة Fuse Setting وهي عبارة عن رابط لموقع خاص يعمل كآلية حاسبة للفيوزات لكل أنواع شرائح الـ AVR كل ما عليك هو الدخول إليه وتحديد الخصائص التي تريدها وسيخبرك الموقع بأي الفيوزات ينبغي لك أن تفعلها وأي منها لا يجب أن تفعلها.



6.7 كيف تعالج الفيوزات المُبرمجة بصورة خاطئة؟

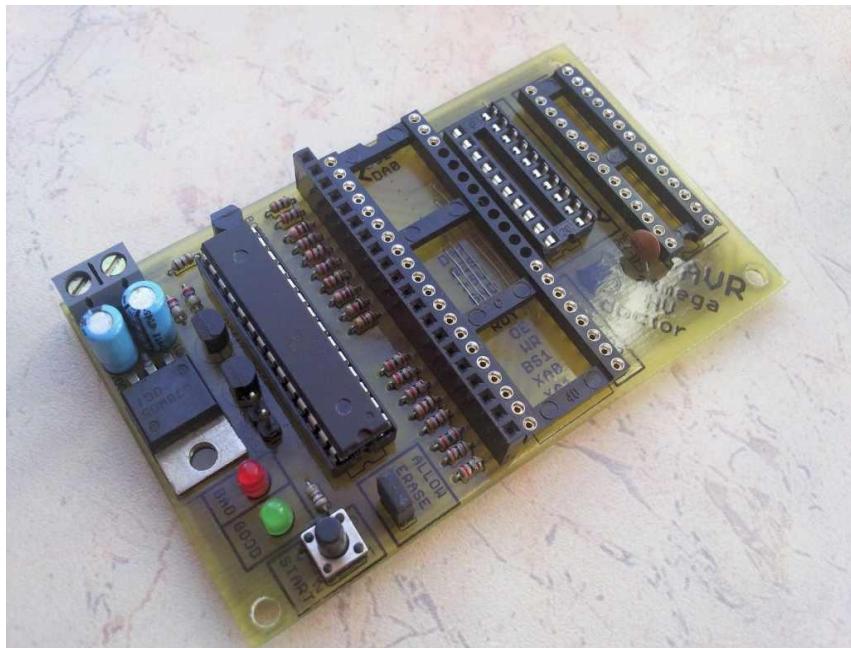
كما ذكرت سابقاً الإعدادات الخاطئة للفيوزات قد تسبب في عدم إمكانية برمجة المُتحكِّم مرة أخرى وبذلك قد تخسر المُتحكِّم، ومع ذلك هناك خبر جيد وهو أنه يتوفّر نوعين من الحلول لهذا الأمر.

الأول: هو شراء و استخدام الـ High Voltage programmers غالبة الثمن مثل AVR Dragon حيث يمتلك القرة على التفقيح والبرمجة عالية الجهد لتصليح الفيوزات.

الثاني: بناء دائرة AVR Fuse Doctor وهي من الدوائر الرائعة التي تستخدم في معالجة الفيوزات بتكلفة منخفضة جداً والتي جربتها بنفسي وكانت رائعة كما أنه يمكنك صناعتها بنفسك بحو 5 دولار فحسب.

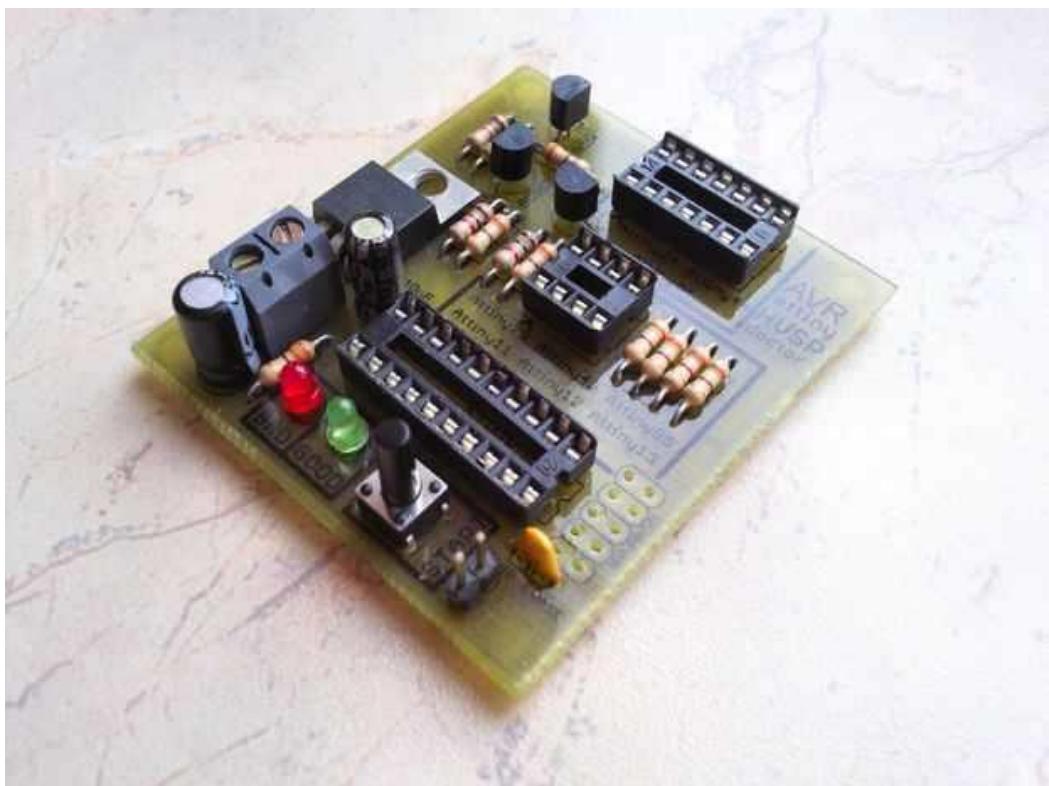
جميع التصميمات لهذه الدائرة يمكنك أن تجدها على الموقع التالي:

<http://mdiy.pl/atmega-fusebit-doctor-hvpp/?lang=en>



تتوفر تصميمات مشابه أيضاً لنفس الدائرة لكن باستخدام مُتحكِّمات أخرى بدل من atmega8 مثل ATTiny Fuse doctor

<http://www.instructables.com/id/AVR-Attiny-fusebit-doctor-HVSP>

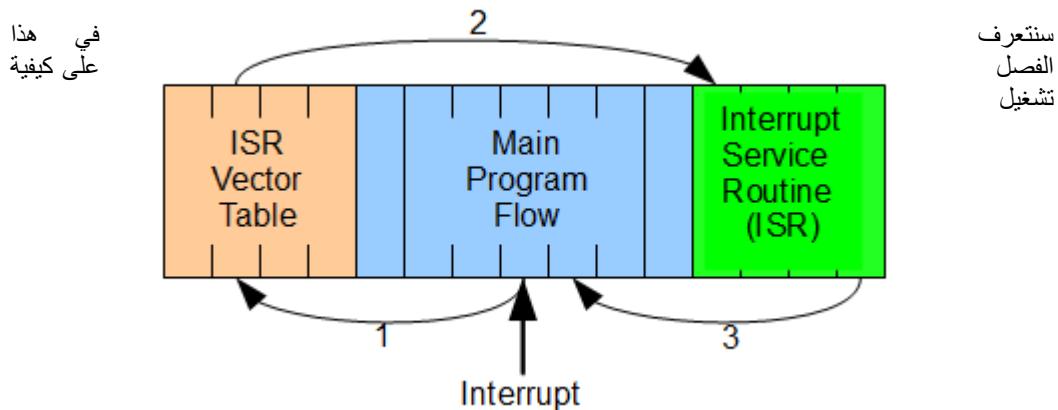




صفحة جديدة



7. المقاطعة Interrupt



المقاطعات الخارجية External Interrupts وفائدة هذه الخاصية الرائعة التي تتيح صناعة تطبيقات ذات استجابة عالية السرعة للأحداث الخارجية.

- ✓ مقدمة عن مفهوم المقاطعة
- ✓ المثال الأول: تشغيل المقاطعة INTO
- ✓ أنواع الإشارات الرقمية Logic, Falling Edge, Raising Edge
- ✓ المثال الثاني: تشغيل INTO مع INT1 مع INT0



7.1 مقدمة عن المُقاطعة The interrupt

في الكثير من الأنظمة المدمجة نجد بعض الوظائف التي تتطلب استجابة فائقة السرعة لحدث معين. لذا تمت مراعاة هذا الأمر في معظم المعالجات والمُتحكمات الدقيقة (حتى القيمة منها) حيث تم إضافة تقنية المُقاطعة interrupt وهي عبارة عن طرف دخل input pin أو حدث برمجي يتسبب في جعل المعالج يتوقف عن ما يفعله الآن ويستجيب للحدث المُسبب للمُقاطعة ويعالجه بسرعة ثم يعود مرة أخرى لما كان يفعله.

مثلاً نجد أن النظام المدمج داخل وحدة تحكم السيارة يعمل على إدارة الوقود وعرض سرعة الحركة ومع ذلك في حالة حدوث اصطدام بجسم ما نجد أن النظام يستجيب بسرعة عالية (بالرغم أنه كان مشغول بمعالجة الوقود والسرعة). تحدث هذه الاستجابة فائقة السرعة بسبب أن الحساسات المسؤولة عن الاصطدام يتم توصيلها على أطراف دخل المُقاطعة وتسمى هذه الأطراف External interrupts.

الحقيقة أنه هناك أنواع كثير للمُقاطعة (بعضها داخلي وبعضها خارجي) سنتحدث في هذا الفصل عن النوع الخارجي فقط External interrupt وسيتم شرح بعض الأنواع الأخرى على مدار الفصول التالية مثل مُقاطعة الـ ADC (في الفصل التالي) و مُقاطعة المؤقتات Timer interrupt.

كيف تعمل المُقاطعة الخارجية

في جميع المعالجات والمُتحكمات الدقيقة يتم تصميم الكود المسؤول عن معالجة المُقاطعات بصورة مستقلة تماماً عن البرنامج الرئيسي main program. فنجد دائماً أن برنامج المُقاطعة ويسمى **Interrupt service routing** (يختصر بكلمة ISR) يكتب في جزء بعيد عن دالة main () فمثلاً يمكنك أن تكتب برنامج main ليقوم بعمل محدد إلى الأبد ثم تكتب برنامج ال ISR ليقوم بوظيفة محددة وسريعة عند تشغيل حساس أو زر معين. يمتلك المُتحكم الدقيق من فئة ATmega16/ATmega32 عدد 3 أطراف للمُقاطعة الخارجية يمكن توصيلها بأي حساس أو مفتاح رقمي وهذه الأطراف هي.

INT0 (pin 2 on port D)

INT1 (pin 3 on port D)

INT2 (pin 2 on port B)

(XCK/T0) PB0	1	40	PA0 (ADC0)
(T1) PB1	2	39	PA1 (ADC1)
(INT2/AIN0) PB2	3	38	PA2 (ADC2)
(OC0/AIN1) PB3	4	37	PA3 (ADC3)
(SS) PB4	5	36	PA4 (ADC4)
(MOSI) PB5	6	35	PA5 (ADC5)
(MISO) PB6	7	34	PA6 (ADC6)
(SCK) PB7	8	33	PA7 (ADC7)
RESET	9	32	AREF
VCC	10	31	GND
GND	11	30	AVCC
XTAL2	12	29	PC7 (TOSC2)
XTAL1	13	28	PC8 (TOSC1)
(RXD) PD0	14	27	PC5 (TDI)
(TXD) PD1	15	26	PC4 (TDO)
(INT0) PD2	16	25	PC3 (TMS)
(INT1) PD3	17	24	PC2 (TCK)
(OC1B) PD4	18	23	PC1 (SDA)
(OC1A) PD5	19	22	PC0 (SCL)
(ICP1) PD6	20	21	PD7 (OC2)



أيضاً يمكن اعتبار الطرف **RESET** أحد أطراف الـ **External interrupt**.

ملحوظة: الـ **RESET** في عالم المُتحكمات الدقيقة يختلف قليلاً عن الحاسب الآلي. فمثلاً نجد في الحاسب الآلي (أو الهاتف الجوال) الزر **RESET** أو الذي يعني **إيقاف تشغيل الطاقة** عن الحاسب ثم إعادة تشغيله. أما في المُتحكمات الدقيقة الزر **RESET** هو مقاطعة خارجية تأمر المعالج أن يترك ما يفعله الآن وينتقل إلى أول أمر في البرنامج المخزن **بداخله** مع تصفيير جميع المُسجلات ووحدات الذاكرة (وهذا يحاكي إعادة تشغيل المُتحكم الدقيق) ومع ذلك تظل الكهرباء متصلة بالمُتحكم ولا يتم فصلها. والسروراء تصميم الـ **RESET** بهذه الطريقة هو أن إعادة فصل وتوصيل الكهرباء بالمُتحكم قد يستغرق 60 ملي ثانية وهذا رقم كبير نسبياً في التطبيقات التي تحتاج استجابة سريعة بينما المقاطعة يتم تشغيلها في أقل من 1 ميكروثانية (يعني أسرع بنحو 60,000 ضعف من فصل الكهرباء وإعادة توصيلها). سنتعرف بالتفصيل على هذا الأمر في الفصل الخاص بدارة الطاقة والفيوزات.

عند إدخال إشارة رقمية على هذه الأطراف تحدث المقاطعة. وعندها يترك المُتحكم الدقيق البرنامج الرئيسي الذي ينفذه وينتقل إلى برنامج الـ **ISR** ليقوم بمعالجة المقاطعة. الكود التالي يمثل التركيب البسيط للـ **ISR** مع الـ **main program** مع الـ **ISR**.

```
int main()
{
    .....
}

ISR(interrupt_type)
{
    .....
}
```

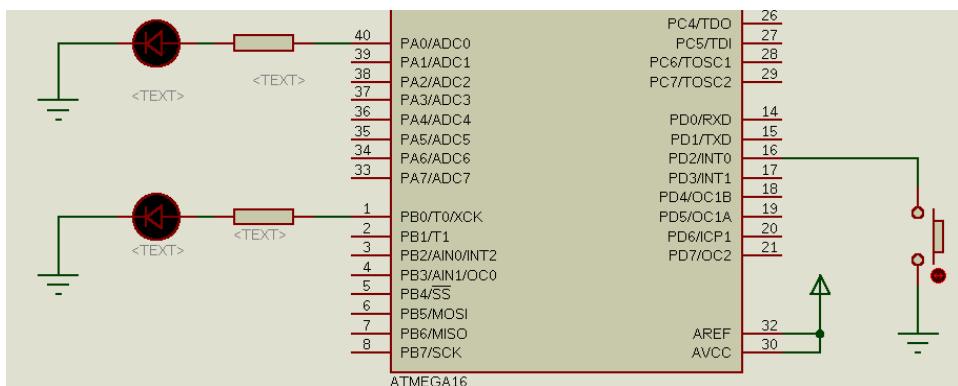
خطوات تفعيل المقاطعة الخارجية

يتم تفعيل المقاطعة الخارجية بمجموعة من الإعدادات كالتالي:

1. ضبط الأطراف التي ستستخدم للمقاطعة مثل **INT0** أو **INT1** لعمل كدخل **input**
2. ضبط نوع الإشارة الكهربائية التي ستنسب المقاطعة على حسب نوع الحساس أو المفتاح الذي سيولد إشارة المقاطعة. (انظر للشرح بالأسفل).
3. يتم تفعيل قبول استقبال المقاطعة على الطرف المطلوب مثل **INT0**
4. تفعيل قبول استقبال المقاطعة بشكل عام
5. كتابة البرنامج الخاص بالمقاطعة **ISR**

7.2 المثال الأول: تشغيل المقاطعة **INT0**

الدائرة التالية عبارة عن 2 دايويد ضوئي + مفتاح الدايويد المتصل بالطرف **PA0** سنقوم بتشغيله بصورة طبيعية ليقوم بعمل **Blink** كل مئة ملي ثانية. أما الدايويد المتصل بالطرف **PC0** سيتم تشغيله أو إطفاؤه فقط عند حدوث مقاطعة على الطرف **.INT0**



الكود

```

#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

int main(void)
{
    // ضبط أطراف التوصيل بالدایودات الضوئية
    DDRA |= (1 << PA0);
    DDRB |= (1 << PB0);

    // ضبط الطرف الخاص بالمقاطعة وتشغيل مقاومة الرفع
    DDRD &= ~(1 << PD2);
    PORTD |= (1 << PD2);

    // ضبط نوع إشارة المقاطة وتفعيل INT0
    MCUCR |= (1 << ISC01);
    GICR |= (1 << INT0);

    // تفعيل قبول المقاطة العامة
    sei();
    while(1)
    {
        PORTA ^= (1 << PA0);
        _delay_ms(100);
    }
}

```

البرنامج الرئيسي



```

return 0;
}

ISR(INT0_vect)
{
    PORTB ^= (1 << PB0);
}

```

برنامـج المقـاطـعة

شرح الكود

في بداية الكود قمنا باستيراد المكتبة المسئولة عن المقاطعات وذلك عن طريق الأمر

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

هذه المكتبة تحتوي على بعض الأوامر الهامة والتي سنستخدمها في الكود. بعد ذلك بدأنا في الدالة main الرئيسية بضبط الأطراف التي سيتصل بها الدايرودات الضوئية وهي الطرفين PA0 و PB0 وذلك عن طريق الأمرين:

```
DDRA |= (1 << PA0);
DDRB |= (1 << PB0);
```

الخطوة التالية كانت ضبط الطرف PD2 ليعمل كدخل وذلك حتى يتمكن من استقبال إشارة المقاطعة من المفتاح المتصل بها. كما قمنا بتشغيل مقاومة الرفع الداخلية internal pull up وذلك حتى نستخدم المفتاح دون الحاجة لتوصيل أي مقاومة إضافية وتم ذلك عن طريق الأمرين:

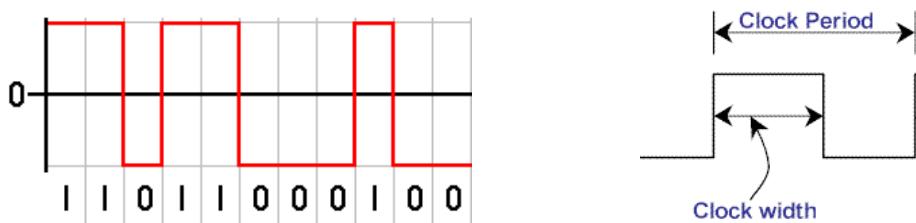
```
DDRD &= ~(1 << PD2);
PORTD |= (1 << PD2);
```

ثم تلى ذلك ضبط نوع المقاطعة الخارجية ونوع الإشارة الكهربائية التي تسبب المقاطعة. وقبل أن نبدأ في شرح الأوامر الخاصة بهذا الأمر علينا أن نتعرّف على بعض الأشياء المتعلقة بالإشارات الكهربائية الرقمية.

الإشارات الرقمية

تنقسم الإشارات الكهربائية الرقمية إلى نوعين و هما **falling or rising Edge** و **logic level**. النوع الأول وهو المعروف لدى الجميع ويسمى LOW أو HIGH أو ويعبر عن القيم الرقمية التقليدية 0 & 1. وتكون كل إشارة سواء 0 أو 1 لها زمان محدد يقاس على حسب الـ **clock** المستخدمة لتشغيل المتحكم الدقيق.

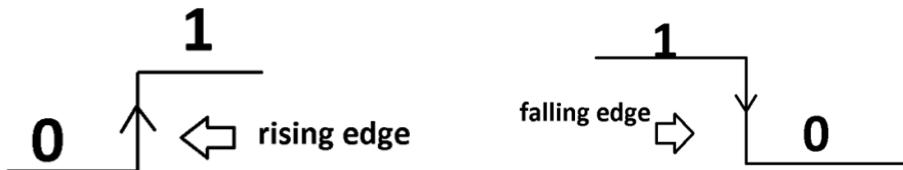
فمثلاً لو كان المتحكم يعمل بـ **clock = 1 Mhz** (مليون هرتز) بـ زمان النبضة الواحد = 1 ميكروثانية ويكون هذا الزمان هو نفس الزمان المطلوب لعمل إشارة كهربائية بقيمة 1 أو صفر. الصور التالية توضح شكل إشارة رقمية مقسمة إلى وحيد وأصفار (حيث يمثل كل مربع رمادي اللون زمان إشارة واحدة).



النوع الثاني من الإشارات الرقمية يسمى "الحـافـة Edges" والتي تنقسم إلى نوعين و هما الحـافـة الصـاعـدة Rising Edge و الحـافـة الـهـابـطـة Falling Edge. هذا النوع من الإشارات الكهربائية يتميز بأنه فائق السـرـعة ولا يلتزم بـ زـمـن مـحدـد و غالباً ما يحدث في زـمـن يـقـاس بـالـنـانـو ثـانـيـة (جزء من مـليـار من الثـانـيـة).

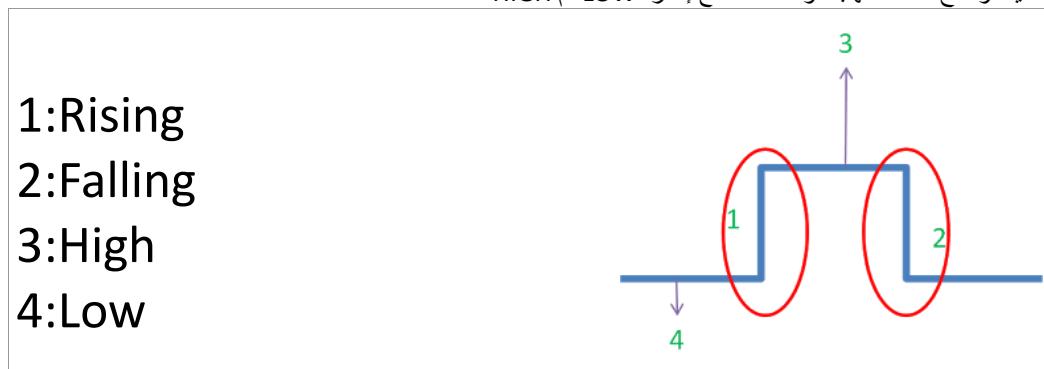


وتعتبر الحواف الصاعدة هي تحول الإشارة الكهربية من LOW level إلى HIGH level في زمن صغير جداً بينما الحواف الهابطة هي تحول الإشارة الكهربية من HIGH level إلى LOW level.



الصور السابقة تثير تساؤل هام وهو: ألا يعني ذلك أن جميع الإشارات الرقمية تحتوي على rising edges و falling edges؟

الإجابة هي نعم. والحقيقة أن أي إشارة رقمية تحتوي على حافة بساوي ضعف عدد الأصفار والواحدات والصورة التالية توضح الحافة الهابط والصاعدة مع إشارة HIGH ثم LOW ثم HIGH.



الأطراف التقليدية التي تعمل كدخل للمتحكم الدقيق لا تستطيع أن تستشعر هذه الحواف سواء كانت الصاعدة أو الهابطة وذلك لأنها تتم في زمن صغير جداً. هنا تظهر مشكلة خطيرة، حيث نجد أن بعض الحواسيب يكون الخرج الكهربائي الخاص بها سريع جداً لدرجة أن الإشارات الكهربائية الناتجة منه تكون في زمن يقاس بالنانو ثانية (مثل الحواف الصاعدة والهابطة) وتنمى هذه الإشارات السريعة بالـ **Electric Edges** أو **Electric Impulse**.

لحل هذه المشكلة قام مصممو المُتحكمات الدقيقة بصناعة دائرة إلكترونية خاصة تتصل بأطراف المقاطعات وتسمى بالـ **edge detector** (مكتشف الحواف). وتكون هذه الدوائر مسؤولة عن الإحساس بالإشارات الكهربائية فائقة السرعة وإبلاغ المتحكم بأنه هناك مقاطعة مطلوبة فوراً.

تمتلك مُتحكمات AVR هذه الدوائر الخاصة على جميع أطراف المقاطعات وبذلك يمكننا أن نجعل المقاطعة تعمل على جميع المفاتيح أو الحساسات فائقة السرعة. حيث يمكن ضبط المقاطعة أن تعمل إما بإشارة تقليدية 1 & 0 أو عن طريق إشارة سريعة **Edge**.

يتحكم في هذا الأمر المُسجل MCUCR والذي يحتوي على مجموعة من البتات مسؤولة عن تحديد نوع إشارة المقاطعة وتنمى **ISCxx** (يتم استبدال xx برقمي 0 و 1 كما سنرى في الشرح التالي). يمكنك الوصول لشرح هذا المُسجل في الصفحة رقم 68 من دليل البيانات للمتحكم ATmega16.



MCU Control Register – MCUCR	The MCU Control Register contains control bits for interrupt sense control and general MCU functions.																																
Bit	<table border="1" style="width: 100%; text-align: center;"> <tr> <td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>SM2</td><td>SE</td><td>SM1</td><td>SM0</td><td>ISC11</td><td>ISC10</td><td>ISC01</td><td>ISC00</td> </tr> <tr> <td>Read/Write</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td> </tr> <tr> <td>Initial Value</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td> </tr> </table>	7	6	5	4	3	2	1	0	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	Read/Write	R/W	Initial Value	0	0	0	0	0	0	0						
7	6	5	4	3	2	1	0																										
SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00																										
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W																										
Initial Value	0	0	0	0	0	0	0																										

- تتحكم البت **ISC00** و **ISC01** في إعدادات الإشارة الخاصة بطرف المقاطةعة **INT0**
- تتحكم البت **ISC10** و **ISC11** في إعدادات الإشارة الخاصة بطرف المقاطةعة **INT1**

عند تغير قيمة هذه البتات يمكننا تحديد نوع إشارة المقاطةعة المطلوبة وذلك تبعاً للجدول التالي (يمكنك أن تجده في الصفحة رقم 68 من دليل بيانات ATmega16)

Table 35. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

الخيار الأول: هو ترك قيمة **ISC00** و **ISC01** بـ0 صفر وهذا سيجعل المقاطةعة تعمل عند إدخال إشارة من نوع **LOW** على الطرف **INT0** (معنى ذلك أنه طالما الطرف **INT0** يدخل إليه إشارة **HIGH** فلن تعمل المقاطةعة).

الخيار الثاني: جعل قيمة **ISC00** تساوي 1 بينما قيمة **ISC01** تساوي صفر وهذا سيجعل المقاطةعة تعمل عند حدوث أي تغير منطقي **logic change** على الطرف **INT0** ومعنى ذلك أنه إذا أدخلت إشارة **HIGH** شرط أن تكون الإشارة السابقة **LOW** أو العكس ستعمل المقاطةعة.

الخيار الثالث: جعل قيمة **ISC00** تساوي 0 بينما قيمة **ISC01** تساوي 1 وهذا سيجعل دائرة **edge detector** تعمل على التقاط أي إشارة كهربائية هابطة وتشغيل المقاطةعة.

الخيار الرابع: جعل قيمة **ISC00** و**ISC01** تساوي 1 وهذا سيجعل دائرة **edge detector** تعمل على التقاط أي إشارة كهربائية صاعدة وتشغيل المقاطةعة.

والآن نعود إلى الكود مرة أخرى سنجد نجد الأمر

MCUCR |= (1 << ISC01);

والذي وضع الرقم 1 داخل البت **ISC01** وبما أن البت **ISC00** تساوي صفر بصورة افتراضية فإن هذا الأمر سيجعل المقاطةعة تعمل عند الحافة الهابطة. وقد اختارت هذا النوع من الإشارات لأن المفتاح المتصل بالطرف **INT0** يعمل مع المقاومة الداخلية لهذا الطرف مما يعني أنه عند الضغط على المفتاح سيتتحول الطرف **INT0** من الحالة **HIGH** إلى الحالة **LOW** وهو ما يوازي الحافة الهابطة. أيضاً كان يمكن أن نترك كلا **ISC00** و **ISC01** بقيمة صفر وهذا سيجعل المقاطةعة تعمل عند الضغط على المفتاح لفترة زمنية قصيرة (1 ميكروثانية على الأقل).

بعد الانتهاء من ضبط نوع إشارة المقاطةعة علينا أن نخبر المتحكم الدقيق بأننا نريد تفعيل استقبال إشارات المقاطةعة على الطرف **INT0** ويتم ذلك عن طريق التلاعيب بالبتات الخاصة بالمسجل **GICR** (يمكنك الوصول لهذا المسجل في الصفحة 69 بدليل البيانات ATmega16).



General Interrupt Control Register – GICR		Bit	7	6	5	4	3	2	1	0	
		Read/Write	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
		Initial Value	0	0	0	0	0	0	0	0	

يمتلك هذا المُسجِّل 3 بُنَات هامة جدًا وهي INT0 و INT1 و INT2 كل بُنَت من الثلَاثة تتحكم في استقبال المُقاطعة على أحد الأطْراف فمثلاً فعندما نضع القيمة 1 داخل البت 0 INT0 فهذا يعني أن المُتحكِّم سيبدأ استقبال إشارات المُقاطعة على الطرف INT0 وهكذا ..

لذا استخدمنا الأمر التالي لأخبار المُتحكِّم أن يبدأ تفعيل المُقاطعة INT0

GICR |= (1 << INT0);

والآن يأتي الأمر

sei();

هذا الأمر مسؤول عن تفعيل استقبال طلبات المُقاطعة بشكل عام للمُتحكِّم الدقيق **وبدون كتابته فلن تعمل أي مُقاطعة مهما كانت** وذلك حتى وإن كنت قد كتبت جميع أوامر ضبط المُقاطعة. ويعتبر استخدام هذا الأمر مع الأمر **cli** أحد الوسائل في التحكم في سير البرامج الهامة التي لا يمكن مقاطعتها.

وأخيرًا نأتي لنهاية الدالة الرئيسية **main** حيث سنقوم بجعل المُتحكِّم يُشعل ويطفئ الدايوه الضوئي المتصل على الطرف PA0 كل 100 ملي ثانية إلى الأبد.

```
while(1)
{
    PORTA ^= (1 << PA0);
    _delay_ms(100);
}
```

بعد الانتهاء من الدالة الرئيسية يأتي دور دالة معالجة المُقاطعة **ISR** حيث نجد أن هذه الدالة مكتوبة بالأسلوب التالي:

```
ISR(INT0_vect)
{
    PORTB ^= (1 << PB0);
}
```

جميع دوال المُقاطعة المختلفة يتم كتابتها عن طريق التعريف **ISR(interrupt_vector)** ويتم استبدال كلمة **interrupt_vector** بنوع المُقاطعة المطلوبة وبالنسبة للـ **External interrupts** هناك 3 أنواع :

INT0_vect
INT1_vect
INT2_vect

فإذا أردنا أن نكتب البرنامج الخاص بالمُقاطعة **INT0** فإننا نكتب

```
ISR(INT0_vect)
{
```

وإذا أردنا أن نكتب البرنامج الخاص بالمُقاطعة **INT1** فإننا نكتب

```
ISR(INT1_vect)
```

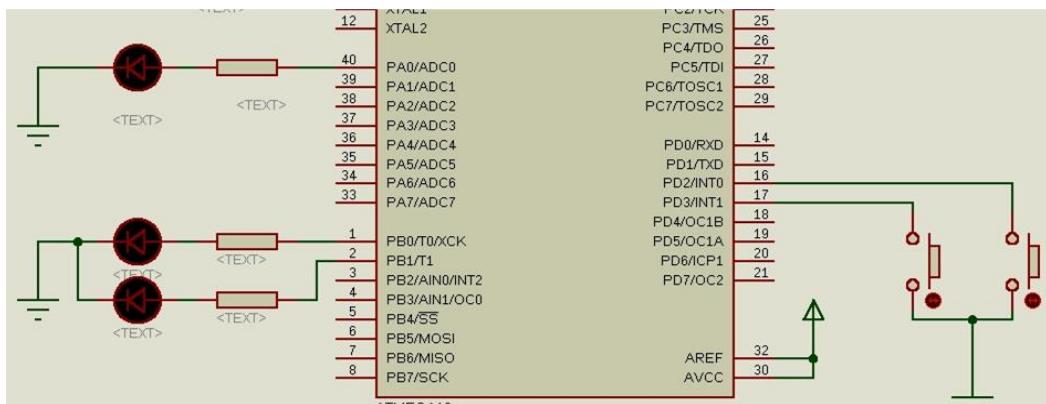


{

مع وضع الأوامر المطلوبة من المقاطعة داخل القوسين {}. وفي المثال السابق استخدام أمر يقوم بعكس حالة الطرف 0 PB0 وهذا ما سيجعل الدايمود الضوئي يضيء أو ينطفئ عند كل مرة يتم فيها تفعيل المقاطعة .INT0.
PORTB ^= (1 << PB0);

7.3 المثال الثاني: تشغيل المقاطعة INT0 مع INT1

في هذا المثال سنقوم بتشغيل كلا المقاطعتين INT0 و INT1 و سنقوم ببناء دائرة مشابهة للمثال السابق باختلاف وجود 2 مفتاح لتفعيل INT0 و INT1 و وجود دايمود ضوئي إضافي كما في الصورة التالية:



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

int main(void)
{
    // ضبط أطراف التوصيل بالدايمودات الضوئية
    DDRA |= (1 << PA0);
    DDRB |= (1 << PB0) | (1 << PB1);
    // ضبط الطرف الخاص بالمقاطعة وتشغيل مقاومة الرفع
    DDRD &= ~((1 << PD2)|(1 << PD3));
    PORTD |= (1 << PD2)|(1 << PD3);
    // تفعيل المقاطعة عند استقبال إشارة من نوع الحافة الهابطة لكل المقاطعتين
    // INT0 & INT1
}
```

البرنامج الرئيسي



```

MCUCR |= (1 << ISC01) | (1 << ISC11);
GICR |= (1 << INT0) | (1 << INT1);

while(1)
{
    PORTA ^= (1 << PA0);
    _delay_ms(100);
}
return 0;
}
// دالة المقاطعة الأولى
ISR(INT0_vect)
{
    PORTB ^= (1 << PB0);
}
// دالة المقاطعة الثانية
ISR(INT1_vect)
{
    PORTB ^= (1 << PB1);
}

```

دوال المقاطعة

شرح الكود

هذا المثال يعتبر مطابق للمثال السابق في نفس الفكرة مع اختلاف تشغيل كلا المقاطعتين INT0 و INT1 بحيث تكون كل مقاطعة مسؤولة عن تشغيل وإطفاء الدايرودات الضوئية المتصلة على الأطراف PB0 و PB1.

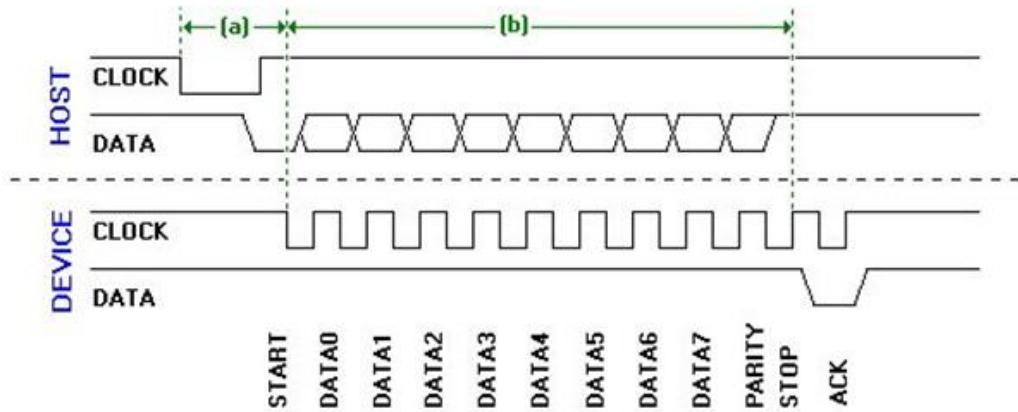
وكما هو ملاحظ ستجد في نهاية البرنامج دالتين ISR الأولى مسؤولة عن أوامر معالجة INT0 و الثانية مسؤولة عن أوامر معالجة INT1.



صفحة جديدة



8. الاتصال التسلسلي بروتوكول UART



في هذا الفصل سنتعرف على أحد أشهر طرق إرسال البيانات بصورة تسلسلية بين المُتحكمات الدقيقة والعالم الخارجي وذلك عبر بروتوكول UART والذي يعتبر أشهر بروتوكول معياري لتبادل البيانات.

- مقدمة عن الاتصال التسلسلي ✓
- الاتصال التسلسلي الغير متزامن ✓
- تهيئة AVR لمحكمات UART ✓
- المثال الأول: تهيئة AVR للعمل كمرسل عبر UART ✓
- المثال الثاني: تهيئة AVR للعمل كمستقبل عبر UART ✓
- المثال الثالث: الإرسال والاستقبال في وقت واحد ✓
- إرسال السلسل النصية Strings ✓
- دوال إضافية ✓



8.1 مقدمة عن الاتصال التسلسلي

عندما يتواصل المتحكم مع العالم الخارجي، فإن إرسال واستقبال البيانات يكون بشكل حزم مكونة من 8 بت (1 بait). بالنسبة لبعض الأجهزة مثل الطابعات القديمة داخل كابل **Parallel port** يتم إرسال البيانات من ناقل البيانات 8 بت (bit data bus-8) من الكمبيوتر إلى ناقل البيانات 8 بت في الطابعة.

يعيب هذا الأسلوب في نقل البيانات وجوب أن تكون المسافة بين الجهازين قصيرة. لأن الأislak تشهـ شـكـ الإـشارـاتـ الكـهـريـةـ معـ طـولـ المـسـافـةـ،ـ كماـ أنـ الأـislakـ المستـخـدمـةـ لـنـقـلـ 8ـ بـتـ فيـ نفسـ الـوقـتـ يـكونـ سـعـرـهـ مـرـفـعـ.

أيضاً تحدث مجموعة من الظواهر كهربائية تسمى "المكثفات الطففية Parasitic Capacitance" و "المكثفات الطففية Parasitic inductance" هذه الظواهر تحدث للوصلات النحاسية المتقاربة من بعضها البعض. وتتسبب في تشويه كبير لشكل الإشارة. الصورة التالية توضح شكل إشارة كهربائية على صورة "نبضة pulse" بعد التشويه.



شكل الاشارة الأصلية

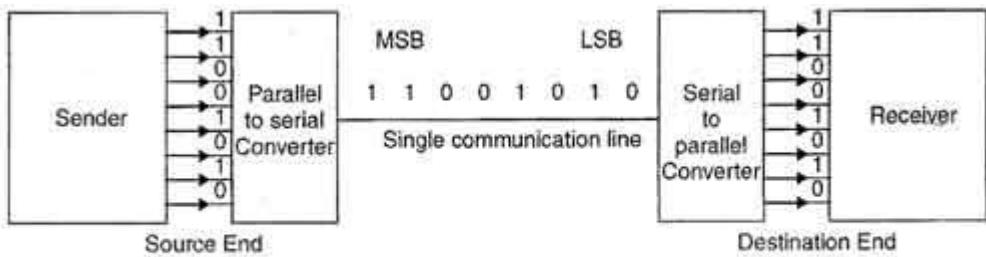
شكل الإشارة بعد التسوية الناتج من المكبات والملفات
الطفولية

لحل هذه المشكلة يتم استخدام الاتصال التسلسلي Serial communication لنقل البيانات بين الأنظمة التي تفصل بينها مسافات كبيرة، وفي وقتنا الحاضر ومع تطور التكنولوجيا أصبح الاتصال التسلسلي أسرع من ذي قبل، فتم تعميمه واستخدامه حالياً في جميع الأجهزة تقريباً بدءاً من الحساسات في الأنظمة المدمجة إلى الحواسيب الشخصية وشبكات الحاسوب الآلي.

مبدأ عمل الاتصال التسلسلي UART

تمتلك المُتحكمات الدقيقة مجموعة من الوسائل التي تمكّنها من توصيل البيانات من وإلى المُتحكم باسلوب تسلسلي ومنها **i2c** – **SPI** – **UART** في هذا الفصل سنتحدث عن الـ **UART**.

تستخدم تقنية الاتصال التسلسلي طرف (سلك) واحد فقط لنقل البيانات من جهاز لآخر بدلاً من 8 أسلاك كما في حالة الاتصال المتوازي Parallel ولكي يتم إرسال البيانات بشكل تسلسلي يتم أو لا تحويل البيانات من 8 بت Parallel إلى 8 باتات متسلسلة وذلك باستخدام شريحة إلكترونية (متواجدة داخل المتحكم الدقيق) تسمى Parallel-in-Serial-out shift register وهو عبارة عن مسجل إزاحة يكون دخله 8 باتات parallel وخرجه 8 باتات متسلسلة. وعلى الجانب الآخر، يجب أن يمتلك المُستقبل شريحة أخرى تقوم بعكس هذه العملية وتسمى Serial-in-Parallel-out shift register، لتحويل البيانات مرة أخرى إلى 8 بت متوازية.



Serial transmission

ملاحظة: كلمة بروتوكول Protocol تعني طريقة تنظيم إرسال واستقبال البيانات مثل سرعة البيانات وطريقة ترتيبها وترقيم البيانات المرسلة وكذلك الأطراف المستخدمة لهذا الإرسال والاستقبال

أنواع الإرسال التسلسلي

يمكن نقل البيانات تسلسلياً ببروتوكول UART بطريقتين، لكل منهما مميزات وعيوب وهما:

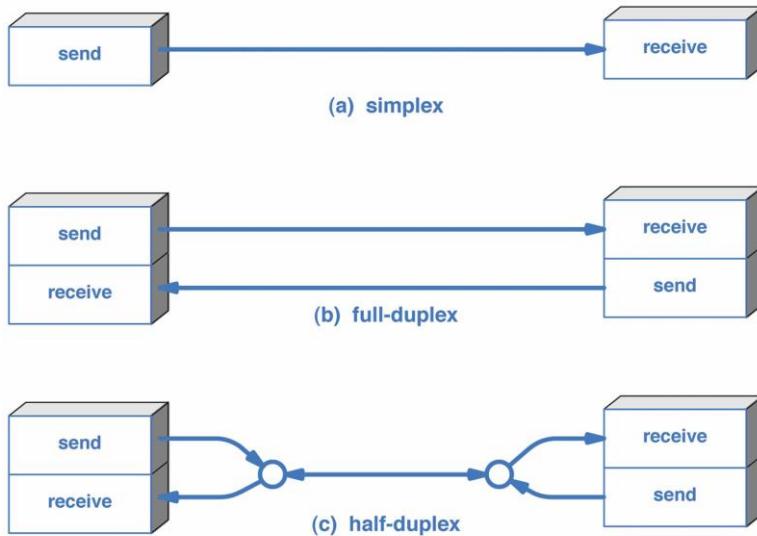
- الاتصال التسلسلي المتزامن.
- الاتصال التسلسلي الغير متزامن.

يُستخدم الاتصال المتزامن لنقل كمية من البيانات دفعة واحدة (Block of data)، بينما يُستخدم الاتصال الغير متزامن لنقل بait واحد في كل مرة.

ويمكن برمجة المُتحكم للعمل بإحدى الطريقتين، ولكن البرنامج سيكون طويلاً. لذلك تم صناعة دوائر متكاملة يتم دمجها داخل المُتحكم مخصصة للاتصال التسلسلي، وأصبح يرمز إليها بـ UART أي Universal Asynchronous Receiver Transmitter. أو USART أي Universal Synchronous-Asynchronous Receiver Transmitter. أو AVR Transmitter. أو AVR Receiver. وتحتوي مُتحكمات AVR على USART داخلي.

هناك نوعان للإرسال التسلسلي:

1. **Simplex**: عندما يكون هناك إرسال فقط أو استقبال فقط. مثل: الطابعة، فالكمبيوتر هو الوحدة الذي يرسل البيانات.
2. **Duplex**: عندما يكون هناك قابلية للإرسال والاستقبال، وينقسم إلى نوعين:
3. **Half-duplex**: وذلك عندما تكون هناك القابلية للإرسال والاستقبال ولكن ليس في آن واحد، مثل: جهاز اللاسلكي، عندما تري التحدث تضغط على الزر وتبدأ في التحدث، والجهاز الآخر يمكنه فقط الاستماع، وعند إزالة يدك من على الزر يمكن للجهاز الآخر إرسال الصوت وأنت يمكنك الاستماع.
4. **Full-duplex**: عندما تكون هناك القابلية للإرسال والاستقبال في آن واحد، مثل: الهاتف المحمول، فبإمكانك التحدث والاستماع لمن تخاطبه بنفس الوقت.

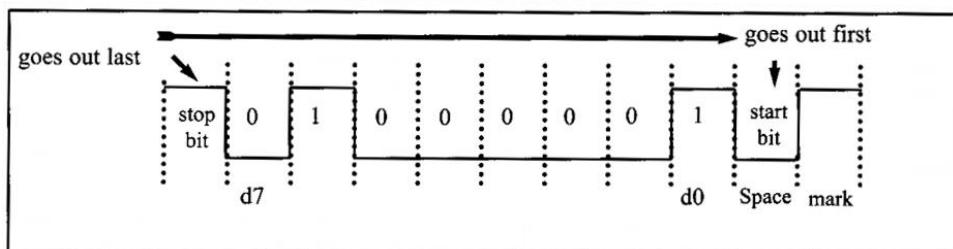


8.2 التسلسلي الغير متزامن Asynchronous

تستقبل البيانات بجهة المستقبل على هيئة 0 و 1. ولا يمكن معرفة ماهية هذه البيانات إلا عندما يتلقى المرسل والمستقبل على مجموعة من القواعد والضوابط "بروتوكول" حول كيفية إرسال البيانات، وكم عدد البيانات في كل مرة، ومتى يبدأ الإرسال ومتى ينتهي.

باتات بداية ونهاية الإرسال:

يتم إرسال البايت الواحد بين بت للبداية **أُخْرَى** للنهاية، وهذا يدعى بالـ "إطار" Frame. بت البداية first bit تكون عبارة عن نبضة واحدة وتكون دائماً LOW، بينما بت النهاية يمكن أن تكون نبضة واحدة أو 2 بت وتكون دائماً HIGH. الصورة التالية تمثل مثال على إرسال كود ASCII للحرف A حيث يتم إرسال 10 بتات لكل 1 بايت. 8 بت للبيانات (حرف A نفسه) وbit للبداية **أُخْرَى** للنهاية.



Courtesy of: AVR microcontroller and embedded systems using assembly and C by M.Ali Mazidi

في بعض الأنظمة تضاف بت **أُخْرَى** وتسمى Parity bit، وتستخدم لمعرفة إذا ما كانت البيانات المستلمة صحيحة أم بها خطأ.



Baud rate: معدل إرسال البيانات

يقاس معدل إرسال البيانات في الاتصال التسلسلي بـ bits per second أي bps أي بـت في الثانية. وتعتمد سرعة إرسال البيانات على النظام المستخدم، فقد كانت أجهزة IBM القديمة ترسل البيانات بسرعات تتراوح بين 100 إلى 9600 bps. ومع التطور استطاعت أجهزة المودم إرسال البيانات بسرعة تصل إلى 56 kbps. في الوقت الحالي تدعم معظم المتحكمات الدقيقة (بما في ذلك AVR) سرعة أنظمة الإرسال التسلسلي من نوع Asynchronous بـ 115200 بـت في الثانية (نحو 100 كيلوبت في الثانية).

أطراف الإرسال والاستقبال في المتحكم ATmega16/32

PDIP

(XCK/T0)	PB0	1	40	PA0 (ADC0)
(T1)	PB1	2	39	PA1 (ADC1)
(INT2/AIN0)	PB2	3	38	PA2 (ADC2)
(OC0/AIN1)	PB3	4	37	PA3 (ADC3)
(SS)	PB4	5	36	PA4 (ADC4)
(MOSI)	PB5	6	35	PA5 (ADC5)
(MISO)	PB6	7	34	PA6 (ADC6)
(SCK)	PB7	8	33	PA7 (ADC7)
RESET		9	32	AREF
VCC		10	31	GND
GND		11	30	AVCC
XTAL2		12	29	PC7 (TOSC2)
XTAL1		13	28	PC6 (TOSC1)
(RXD)	PD0	14	27	PC5 (TDI)
(TXD)	PD1	15	26	PC4 (TDO)
(INT0)	PD2	16	25	PC3 (TMS)
(INT1)	PD3	17	24	PC2 (TCK)
(OC1B)	PD4	18	23	PC1 (SDA)
(OC1A)	PD5	19	22	PC0 (SCL)
(ICP1)	PD6	20	21	PD7 (OC2)

الطرف الذي تحمل الرمز RXD تستخدم للاستقبال: ويتم توصيلها بالطرف الخاصة بالإرسال في المتحكم الآخر.
الطرف الذي تحمل الرمز TXD تستخدم للإرسال: ويتم توصيلها بالطرف الخاصة بالاستقبال في المتحكم الآخر.

8.3 تهيئة الـ UART الداخلي لمتحكمات AVR

يتم تهيئة الـ UART للعمل عن طريق ضبط الإعدادات الخاصة بـ: معدل نقل البيانات – عدد برات الإرسال – عدد برات



النهاية... وغيرها من الإعدادات والتي يتم ضبطها عن طريق تغيير قيم المسجلات التي تحكم في ال UART. يتحكم في ال 5 UART مسجلات وهي:

- .UBRR [H: L]: USART Baud Rate Register -1
- .UCSRA: USART Control and Status Register A -2
- .UCSRB: USART Control and Status Register B -3
- .UCSRC: USART Control and Status Register C -4
- .UDR: USART I/O Data Register -5

شرح المسجلات

UBRR [H: L]

15	14	13	12	11	10	9	8	UBRRH
URSEL	-	-	-	UBRR[11:8]				UBRRL
7	6	5	4	3	2	1	0	UBRRL

وهو عبارة عن مسجلين 8 بت UBRRL ويحمل ال 8 بت 1 بت القيمة الصغرى من قيمة ال baud rate ، والمسجل الآخر هو UBRRH ويحتوى على القيمة المطمى من ال baud rate يتم وضع قيمة ال baud rate في البات من 0 إلى 11 . ملاحظة: بالنسبة لبعض المسجلات سيكون الشرح متعلق بالباتات التي سنسخدمها فقط

UCSRA

7	6	5	4	3	2	1	0	UCSRA
RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	R/W

- **البت رقم 7: RXC** هذه البت تصبح 1 عند اكتمال استقبال البايت ، وتظل 0 أثناء الاستقبال.
- **البت رقم 6: TXC** هذه البت تصبح 1 عند تمام الإرسال ، وتظل 0 أثناء الإرسال.
- **البت رقم 5: UDRE** تكون قيمتها 0 أثناء انشغال المتحكم وتصبح 1 عندما يكون جاهزاً لإرسال بيانات أخرى.

UCSRB

7	6	5	4	3	2	1	0	UCSRB
RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	R/W

- **البت رقم 7: RXCIE** عند جعل قيمة هذه البت تساوي 1 ، يتم تفعيل المقاطعة Interrupt الخاصة باستقبال البيانات.
- **البت رقم 6: TXCIE** عند جعل قيمة هذه البت تساوي 1 ، يتم تفعيل المقاطعة Interrupt الخاصة بارسال البيانات.
- **البت رقم 5: UDRIE** عند جعل قيمة هذه البت تساوي 1 ، يتم تفعيل المقاطعة Interrupt الخاصة بجاهزية المتحكم لإرسال أو إستقبال البيانات.
- **البت رقم 4: RXEN** عند جعل قيمة هذه البت تساوي 1 يتم تفعيل إمكانية استقبال البيانات.
- **البت رقم 3: TXEN** عند جعل قيمة هذه البت تساوي 1 يتم تفعيل إمكانية إرسال البيانات.



البت رقم 2: UCSZ2 برجاء مراجعة الجدول في الصفحة التالية

• **UCSRC**

7	6	5	4	3	2	1	0	UCSRC
URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

يحتوي هذا المسجل على 2 بت لهما أهمية قصوى وهم البت رقم 2: UCSZ1 وكذلك البت رقم 1: UCSZ0 حيث يستخدمان في تحديد عدد برات الإرسال في حزمة البيانات الواحدة.

الجدول التالي يوضح كيفية ضبط حجم الحزمة الواحدة من البيانات وذلك بتغيير قيم هذه البتات.

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

ستستخدم في الأمثلة التالية نظام الإرسال بحجم 8 بت (1 بait). وهذا النظام يعتبر قياسي في معظم المُتحكمات الدقيقة والأجهزة الإلكترونية المختلفة.

8.4 المثال الأول: تهيئة الـUART للعمل كمرسل

نسترجع ما شرحناه سابقاً عن بروتوكول الاتصال التسلسلي، وكما أشرنا فإن هناك ما يسمى بمعدل إرسال البيانات والذي يجب أن يتم تحديده للمتحكم، وأيضاً يجب تحديد عدد البتات التي سنرسلها في المرة الواحدة، فالمتحكم قادر على إرسال 5،7،8 أو 9 برات في المرة الواحدة، هذا بخلاف نبضة البداية ونبضة النهاية. ولكن اتفقنا على اتباع الأنظمة القياسية في تحديد عدد 8 برات للإرسال، وهذا ما يجب تحديده للمتحكم.

نبدأ أولاً بتحديد معدل نقل البيانات baud rate ويتم تخزين القيمة في المسجلين UBRRH و UBRRRL. لذا نأخذ على سبيل المثال معدل إرسال بيانات يساوي 9600 bps. بمراجعة دليل البيانات للمتحكم يتضح أن تحديد القيمة التي يجب تخزينها بالمسجلين UBRR[H:L] يتم عن طريق العلاقة:

$$UBRR = \frac{f_{OSC}}{16BAUD} - 1$$

• Fosc هو تردد المذبذب الداخلي أو الـ "Crystal" وسنفترض أننا نشغل المتحكم بتردد 16 ميجا هرتز.

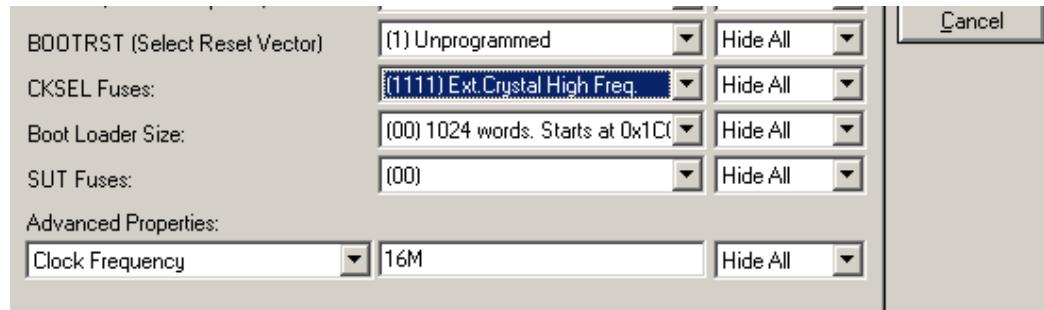
• BAUD هي قيمة معدل إرسال البيانات والتي حددناها سلفاً بـ 9600 bps.



ملاحظة: في هذا المثال سيتم تشغيل المتحكم بسرعة 16 ميجا وذلك عبر توصيل كريستال خارجية 16 ميجا + مكثفات 22 بيكوفاراد، يمكنك مراجعة فصل الطاقة وسرعة التشغيل للتعرف أكثر على خواص هذا النوع من المذبذبات وطريقه عمله. ولا تنسى أن تضبط برنامج بروتوكول علىمحاكاة المتحكم بسرعة تشغيل 16 ميجا وذلك عبر الضغط على رمز شريحة ATmega16 مرتين ثم تغير الكريستال وقيمتها

بالتعويض عن هذه القيمة في المعادلة السابقة تكون قيمة UBRR تساوي 103.16667 أي بالتقريب تساوي 103. ويتم وضع هذه القيمة كما هو موضح في الكود التالي (هذا الكود يجعل المتحكم يرسل قيمة الحرف A بصيغة ASCII كل ثانية).

#define F_CPU 16000000



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

```
int main(void)
{
    uint16_t UBRR_Value = 103;
    UBRRRL = (uint8_t) UBRR_Value;
    UBRRRH = (uint8_t) (UBRR_Value >> 8);
    UCSRB = (1<<RXEN) | (1<<TXEN);
    UCSRC |= (3<<UCSZ0);

    while(1)
    {
        while( ! (UCSRA & (1<<UDRE) ) );
        UDR = 'A';
        _delay_ms(1000);
    }

    return 0;
}
```

شرح الكود

بدايةً قمنا بتعريف متغير 16 بت اسمه UBRR_Value لتخزين القيمة المطلوب كتابتها في المسجلين [H:L]. ثم أمرنا المتحكم بتخزين هذه القيمة في المسجل UBRRRL ولكن هذا المسجل 8 بت فقط. حيث سيتم تخزين أول 8 بت فقط من القيمة. ثم قمنا بتخزين باقي البتات في المسجل UBRRRH عن طريق الأمر



UBRRH = (unsigned char) (UBRR_Value >> 8);

وهذا الأمر يقوم بعمل إزاحة لليمين بمقدار 8 بت. ويخزن باقي البتات في هذا المُسجل.

محتوى المتغير :UBRR_Value

0	0	0	0	0	0	0	0	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ما تم تخزينه بالمسجل UBRRRL :

0	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---

ما تم تخزينه بالمسجل UBRRH بعد عمل إزاحة لليمين بمقدار 8 بت:

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

إلى هنا انتهينا من تحديد قيمة ال baud rate . نأتي الأن لتعديل إمكانية الإرسال والاستقبال عن طريق الأمر التالي:

UCSRB = (1 << RXEN) | (1 << TXEN);

بعد هذا الأمر يتبقى شيء واحد وهو تحديد عدد البتات المرسلة في المرة الواحدة.

UCSRC |= (3 << UCSZ0) ;

وهذا الأمر يقوم بتعيين عددهم إلى 8 بتات. وهو مساوي للأمر

UCSRC |= ((1 << UCSZ2) | (1 << UCSZ0));

والذي يقوم بوضع القيمة 1 في كلا من UCSZ0 & UCSZ2 . ولكن للتسهيل استخدمت الأمر بصورته الأولى. بذلك تكون قد انتهينا من تهيئة ال UART ونستطيع أن نرسل البيانات. ولكن لكي نبدأ الإرسال يجب أن نضع هذه البيانات في المُسجل UDR وكما ذكرنا سابقاً، يجب أن ننتظر حتى يصبح المُتحكم جاهزاً لإرسال البيانات لذلك استعينا بالأمر التالي:

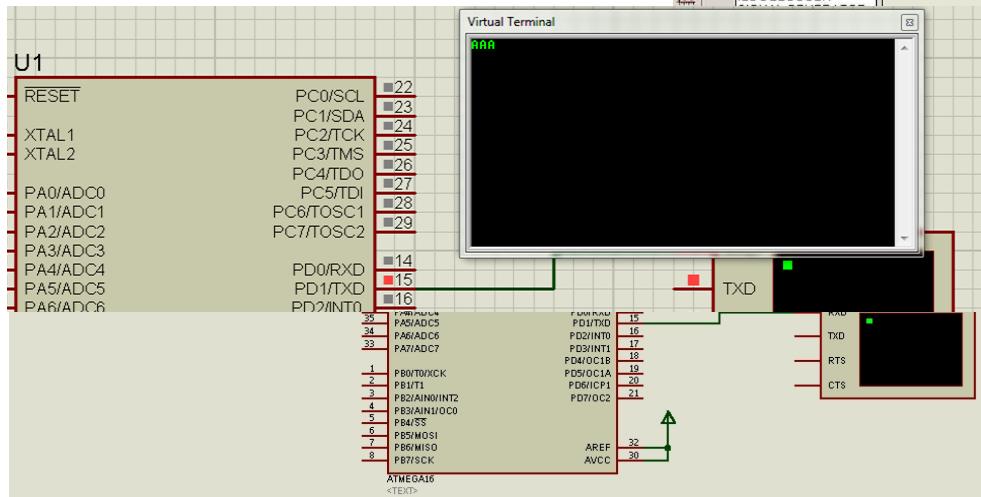
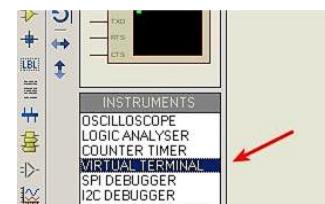
While (! (UCSRA & (1 << UDRE)));

ويعنيه أن ينتظر المُتحكم دون فعل أي شيء طالما البت رقم 5 في المُسجل UCSRA لا تساوي 1. كما ذكرنا سابقاً عند شرح المُسجلات أن وجود 1 في هذه البت يدل على أن المُتحكم جاهز لإرسال البيانات.

الصورة التالية توضح دائرة محاكاة الكود على برنامج بروتس. مع العلم أنه

يمكن محاكاة ال serial port وذلك باستخدام الأداة virtual terminal

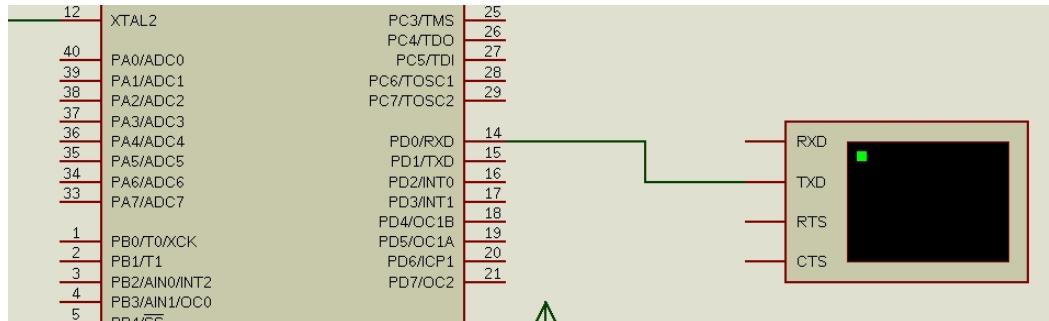
وها هو ذا الحرف A يتم إرساله كل ثانية عند تشغيل المحاكاة:





8.5 المثال الثاني: تهيئة الـ UART للعمل كمستقبل

استقبال البيانات عن طريق الـ UART يتم بنفس الكود مع عمل تغييرات بسيطة في الدائرة واضافة سطر جديد.(لاحظ أنه في الدائرة الجديدة يتم توصيل الطرف TXD في الـ virtual terminal بالطرف RXD في المتحكم RXD في المتحقق الدقيق).



```

#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    uint16_t UBRR_Value = 103;
    UBRRRL = (uint8_t) UBRR_Value;
    UBRRRH = (uint8_t) (UBRR_Value >> 8);
    UCSRB = (1<<RXEN) | (1<<TXEN);
    UCSRC |= (3<<UCSZ0);

    while(1)
    {
        while (! (UCSRA & (1 << RXC)));
        PORTC = UDR;
    }

    return 0;
}

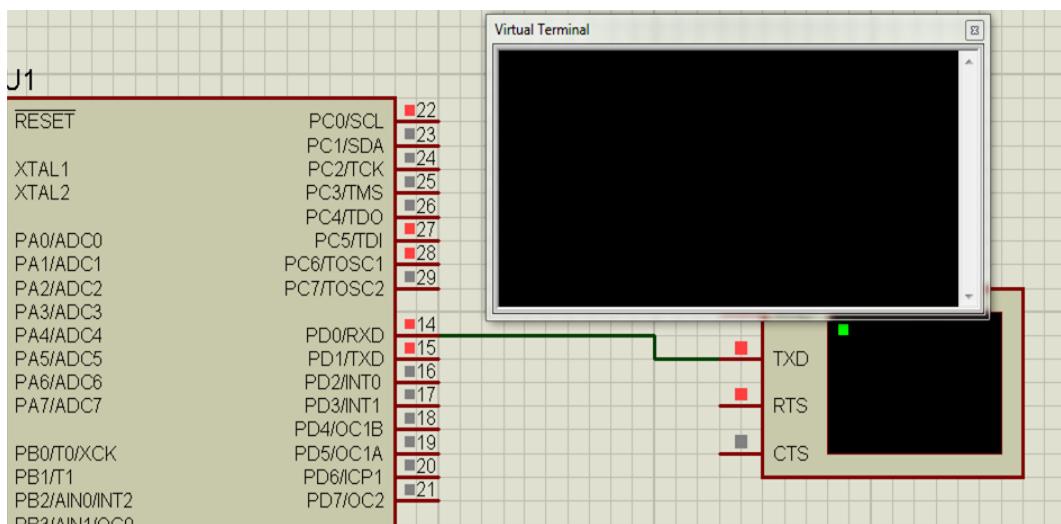
```

الاختلاف الوحيد في الكود نجده في الأمر التالي:

While (! (UCSRA & (1 << RXC))); // Waiting for Receiving buffer to be empty.

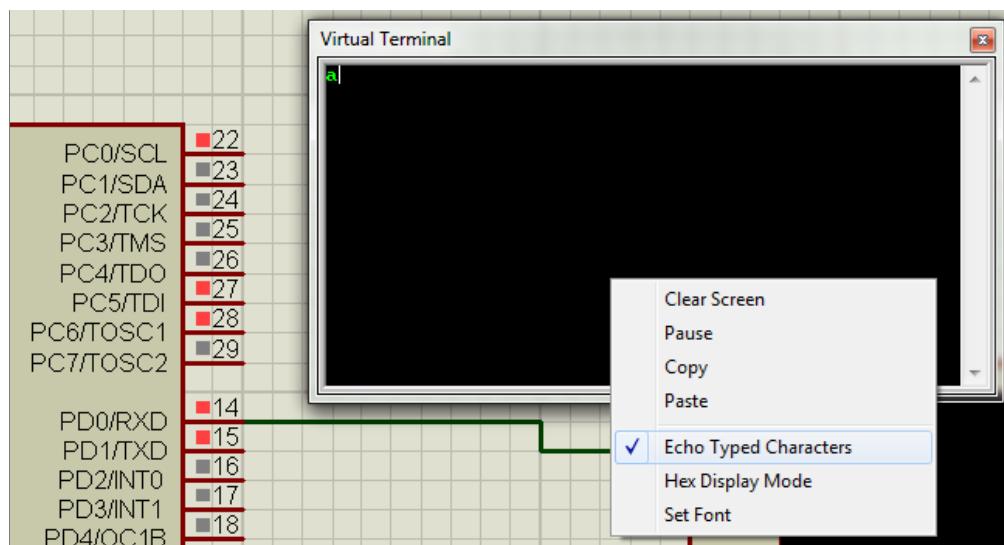
وهذا معناه الانتظار حتى يصبح المتحكم جاهزاً للاستقبال. والأمر الذي يليه يقوم بعرض قيمة ما تمت طباعته في نافذة .PORTC على Virtual terminal

شكل التجربة أثناء استقبال الحرف a وكذلك إخراج قيمته (b011000010) على .PORTC



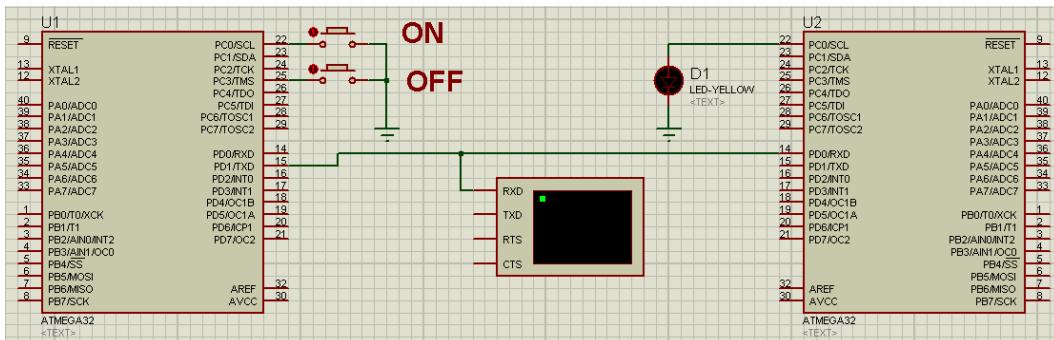
ملاحظة: أثناء كتابة أي حرف على نافذة Virtual terminal لا يتم طباعته على الشاشة ولكن يتم ارساله، وإذا أردت أن يتم طباعته للتحقق مما تضيعه عليه، فكل ما عليكم فعله هو الضغط بالزر الأيمن للماوس على النافذة واختيار Echo .

Typed Characters



8.6 المثال الثالث: الإرسال والاستقبال في وقت واحد

والآن، ما رأيك أن ندمج المثالين السابقين في برنامج واحد. نرسل حرف من متحكم آخر، وعند استقبال هذا الحرف يضي المتحكم الآخر الديود الضوئي المتصل به.



لاحظ توصيل الطرف TXD في المُتحكّم بالجانب الأيسر إلى الطرف RXD في المُتحكّم الموجود بالجانب الأيمن.

أيضاً يستخدم زرين، الأول ومكتوب بجانبه ON سيقوم بإرسال الحرف " N "، وعند استقبال هذا الحرف من قبل المتحكم الثاني سيقوم بإضاعة الدايرود الضوئي. أما الثاني ومكتوب بجانبه OFF فسيقوم بإرسال الحرف " F "، وعند استقباله من قبل المتحكم الثاني سيقوم بإطفاء الدايرود الضوئي.

هذا هو الكود الخاص بالمحكم الأول والذي يتولى مهمة إرسال الأحرف عند الضغط على أي من الزررين.

```
#define F_CPU 16000000  
#include <avr/io.h>  
#include <util/delay.h>
```

```

int main(void)
{
    DDRC &= ~((1<<PC0) | (1<<PC3));           // ضبط الأطراف لتعمل كدخل
    PORTC |= (1<<PC0) | (1<<PC3);           // تفعيل مقاومة الرفع

    uint16_t UBRR_Value = 103;

    UBRRL = (uint8_t) UBRR_Value;
    UBRRH = (uint8_t) (UBRR_Value >> 8);

    UCSRB = (1<<RXEN) | (1<<TXEN);

    UCSRC |= (3<<UCSZ0);

    while(1)
    {
        if(bit_is_clear(PINC,0))
        {
            while(!(UCSRA & (1<<UDRE)));
            UDR = 'N';
            _delay_ms(300);
        }
    }
}

```



```

if(bit_is_clear(PINC,3))
{
    while(!(UCSRA & (1<<UDRE)));
    UDR = 'F';
    _delay_ms(300);
}
return 0;
}

```

السطر الأول يقوم بتحديد الأطراف PC0 و PC3 كمدخل رقمية. من خلال إدخال القيمة 0 في البات المناهضة لهما في المُسجل DDRC. والسطر الذي يليه يقوم بتفعيل مقاومة الرفع الداخلية لكل منهما.

ثم يأتي الجملة الشرطية **((0,if(bit_is_clear(PINC,3)))** هذا السطر يقوم باختبار ما إذا تم الضغط على الزر المتصل ب PC0 أم لا، فإذا تم الضغط على الزر سيقوم المُتحكم بارسال الحرف "N". وكذلك الأمر بالنسبة للزر المتصل ب PC3 ولكن مع فارق أنه يقوم بإرسال الحرف "F".

أما الكود الخاص بالمُتحكم المسئول عن استقبال الأحرف وإضاعة أو إطفاء الدايدون الضوئي فهو كالتالي:

```

#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRC |= (1<<PC0);
    uint16_t UBRR_Value = 103;
    char Received;

    UBRR0L = (uint8_t) UBRR_Value;
    UBRR0H = (uint8_t) (UBRR_Value >> 8);
    UCSRB = (1<<RXEN) | (1<<TXEN);
    UCSRC |= (3<<UCSZ0);

    while(1)
    {
        while (! (UCSRA & (1 << RXC)));
        Received = UDR;

        if(Received == 'N')
            PORTC |= (1<<PC0);
        if(Received == 'F')
            PORTC &= ~(1<<PC0);
    }
}

```



```

    }
    return 0;
}

```

شرح الكود

في هذا البرنامج قمنا بإنشاء متغير من نوع Character ويدعى Received وقمنا بتخزين ما يتم استقباله في هذا المتغير، ثم يقوم المتحكم باختبار محتوى هذا المتغير بجملتين شرطيتين، فإذا كان محتواه مساوياً للحرف "N" قام بإضافة الدايويد الضوئي المتصل بPC0، وإذا كان محتواه مساوياً للحرف "F" قام بإطفاء الدايويد الضوئي.

8.7 إرسال مجموعة بيانات مثل السلسل النصية

قد يتادر إلى ذهنك، ماذا أفعل إذا أردت إرسال كلمة أو جملة؟ ماذا أفعل إذا أردت إرسال قيمة متغير؟ وماذا أفعل لكي استطع إستقبال كلمة أو جملة؟ لنفترض أننا نحاول إرسال كلمة "UART" سنجد أنه هناك طريقتين لذلك.

الطريقة الأولى: إرسال حروف متتالية

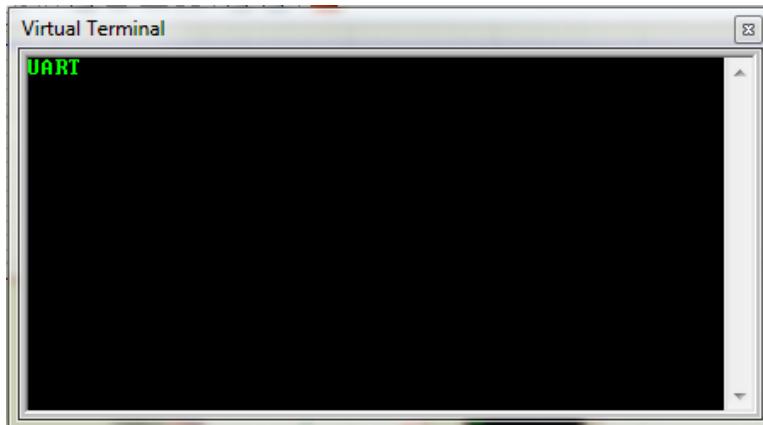
```

while(!(UCSRA & (1<<UDRE)));
    UDR = 'U';                                // إرسال حرف U
while(!(UCSRA & (1<<UDRE)));
    UDR = 'A';                                // إرسال حرف A
while(!(UCSRA & (1<<UDRE)));
    UDR = 'R';                                // إرسال حرف R
while(!(UCSRA & (1<<UDRE)));
    UDR = 'T';                                // إرسال حرف T

```

وسيتم إرسالها.

هذه الطريقة
باتكيد لكنها
وتتطلب
من الأكواذ
(ما يعني)
المزيد من
ROM
الدقيق) كما
العبارات
سيؤدي ذلك
ضخم



بالرغم أن
تصلح
ليست فعالة
كتابة الكثير
البرمجية
استهلاك
ذاكرة الـ
للمتحكم
أنه في حالة
الطويلة
لاستهلاك
لذاكرة.

الطريقة الثانية: استخدام المؤشرات

ولكن هناك طريقة أخرى. وهي استخدام ال Pointer فالكلمة مكونة من عدد من الأحرف بجانب بعضها. لذا يمكن استبدال الكود السابق بهذا الكود.

```
char *word = "UART";
```



```

while(*word > 0)
{
    while(!(UCSRA & (1<<UDRE)));
    UDR = *word++;
}

```

شرح الكود

في هذا الكود استخدمنا مؤشر يشير إلى بداية الكلمة. ومن أساسيات علم الكمبيوتر أن أي String يحتوى آخره على الرقم 0 ويدعى "null character" لذلك استخدمنا الحلقة التكرارية while(*word > 0) أي طالما أنه يشير إلى شيء أكبر من الـ 0 ستستمر الحلقة بالتكرار.

الأمر *word++ يقوم بإرسال الحرف الذي يشير إليه المؤشر حالياً، ثم يقوم بزيادة المؤشر ليشير الحرف التالي، حتى يصل إلى نهاية الكلمة فيشير إلى الرقم 0 الذي يتواجد بنهائية أي String، فلا يتحقق شرط الحلقة التكرارية وينتهي تنفيذها.

حسناً، إلى هنا أردنا استخدام الـ UART أصبح لزاماً علينا كتابة الأسطر الخاصة بتهيئة الـ UART وأيضاً عند إرسال حرف أو كلمة، يجب كتابة الأسطر الخاصة بذلك، ولكن ما رأيك يجعل الكود أكثر قابلية للاستخدام المتكرر.

ل فعل ذلك يجب علينا أن نستخدم الدوال، لأنها تسهل الأمر كثيراً. ونضع بداخل كل دالة مجموعة الأوامر التي ستتلقاها هذه الدالة. مثال على ذلك. الدالة الخاصة بتهيئة الـ UART.

```

void UART_init()
{
    uint16_t UBRR_Value = 103;
    UBRRRL = (uint8_t) UBRR_Value;
    UBRRRH = (uint8_t) (UBRR_Value >> 8);
    UCSRB = (1<<RXEN) | (1<<TXEN);
    UCSRC |= (3<<UCSZ0);
}

```

نكتب الدالة السابقة قبل دالة main. وبداخل دالة ال main نقوم باستدعاءها لتنفيذ الأوامر التي بداخلها عن طريق الأمر التالي:

UART_init();

كما هو موضح في الكود التالي (إعادة كتابة المثال الأول ولكن بصورة أفضل):

```

#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>

void UART_init()
{
    uint16_t UBRR_Value = 103;
    UBRRRL = (uint8_t) UBRR_Value;
    UBRRRH = (uint8_t) (UBRR_Value >> 8);
    UCSRB = (1<<RXEN) | (1<<TXEN);
    UCSRC |= (3<<UCSZ0);
}

```



```

int main(void)
{
    UART_init() ;

    while(1)
    {
        while( ! (UCSRA & (1<<UDRE) ) );
        UDR = 'A';
        _delay_ms(1000);
    }

    return 0;
}

```

والآن استرجع ما ذكرناه في شرح بروتوكول ال UART عن معدل نقل البيانات، وأن هذه السرعة يمكن أن تتغير، أليس من الأفضل جعل الدالة **UART_init** بإمكانها أن تحدد سرعة نقل البيانات عن طريق أن نمرر هذه السرعة إلى الدالة عند استدعائهما؟

تذكر أنه يمكن تحديد هذه السرعة من العلاقة السابق ذكرها،

$$UBRR = \frac{f_{OSC}}{16BAUD} - 1$$

إذاً يمكننا كتابة المعادلة التالية لإيجاد القيمة المراد تخزينها داخل المُسْجِل UBRR:

```
uint16_t UBRR_Value = lrint ( (F_CPU / (16L * baud_rate) ) -1);
```

حيث:

F_CPU: تردد المذبذب الذي يعمل عليه المُتَحَكِّم.

Baud_rate: متغير ندخل به قيمة سرعة إرسال البيانات مثل: 9600.

هي دالة تقوم بتقريب الناتج إلى أقرب رقم صحيح، ولاستدعاؤها لابد من تضمينها باستخدام **include** ثم اسم الملف **math.h** في بداية البرنامج.

فيصبح شكل الدالة كالتالي:

```

#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>
#include <math.h> // استدعاء مكتبة الحساب

void UART_init()
{
    uint16_t UBRR_Value = lrint ( (F_CPU / (16L * baud_rate) ) -1);
    UBRRRL = (uint8_t) UBRR_Value;
    UBRRRH = (uint8_t) (UBRR_Value >> 8);
    UCSRB = (1<<RXEN) | (1<<TXEN);
    UCSRC |= (3<<UCSZ0);
}

```



```
}
```

ويتم استدعائهما بداخل ال main بالشكل التالي:

```
int main(void)
{
    UART_init(9600) ;
    ....
```

دوال إضافية 8.8

أليس هذا أسهل من ذي قبل؟ إذًا هيا لنفعل المثل ونكتب دالة خاصة بإرسال حرف، وأُخرى خاصة بإرسال كلمة أو جملة.
وأُخرى خاصة باستقبال حرف وواحدة خاصة باستقبال كلمة أو جملة.

الدالة الخاصة بإرسال حرف

```
void UART_send_char(char data)
{
    while( !(UCSRA & (1<<UDRE) ) );
        UDR = data;
}
```

الدالة الخاصة باستقبال حرف

```
char UART_receive_char()
{
    while ( !(UCSRA & (1 << RXC) ) );
        return UDR;
}
```

الدالة الخاصة بإرسال String سلسلة حروف

(لاحظ أن هذه الدالة تعتمد على دالة إرسال حرف واحد).

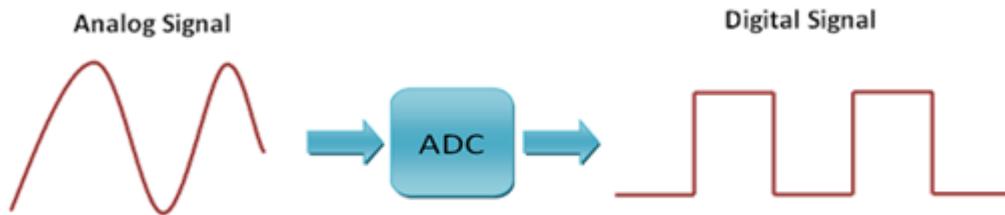
```
void UART_send_string(char *data)
{
    while(*data > 0)
        UART_send_char(*data++);
        UART_send_char('\0');
}
```



صفحة جديدة.....



9. المُحوَّل التَّنَاظُرِي-الرَّقْمِي ADC



في هذا الفصل سنتعرف على كيفية قراءة الجهد الكهربائي المتغيرة Analog وتحويلها إلى قيم رقمية وذلك باستخدام المحوَّل التَّنَاظُرِي-الرَّقْمِي المدمج داخل مُتَحَكِّمات AVR. حيث يمكن استغلال هذا المحوَّل في قراءة الحسَّاسات التَّنَاظُرِيَّة أو أي عَنْصُر إلْكْتَرُونِيٍّ له خُرُجٌ كَهْرَبَيٌّ متغِيرٌ.

- ✓ مقدمة عن المحوَّل التَّنَاظُرِي - الرَّقْمِي ADC
- ✓ طريقة عمل المحوَّل ADC
- ✓ تركيب المحوَّل ADC داخل المُتَحَكِّم ATmega16
- ✓ مثال: قراءة جهد متغير باستخدام مقاومة متغيرة
- ✓ الحسابات الرياضية الخاصة بالـ ADC



9.1 مقدمة عن المحول التناضري-الرقمي ADC

الإشارات الكهربائية في العالم الخارجي ليست مقتصرة فقط على الإشارات الرقمية Digital بل على العكس العديد من الأجهزة و الحساسات الإلكترونية تصدر إشارات تماثيلية Analog فمثلاً نجد أن معظم الحساسات المتوفرة في الأسواق يمكنها تحويل كمية فизيائية معينة مثل (درجة الحرارة - الرطوبة - ضغط - تركيز مادة كيميائية .. الخ) إلى فرق جهد كهربائي يتغير بتغير هذه القيم الفيزيائية.

بصورة طبيعية لا تستطيع الحواسيب أو المُتحكمات الدقيقة أن تفهم الإشارات التماثيلية فكل ما تفهمه هذه الحواسيب هي الإشارات الرقمية مثل 1 و 0 أو HIGH و LOW. لذا قام مصممو الإلكترونيات بصناعة شريحة خاصة تسمى بالمحول التناضري الرقمي **Analog to Digital Converter** و تختصر بكلمة **ADC** هذه الشريحة يمكن شرائها بصورة مستقلة و توصيلها مع المُتحكم الدقيق.

لحسن حظنا نجد أن مصممي مُتحكمات AVR قاموا بدمج هذه الشريحة بصورة جاهزة للعمل على معظم مُتحكمات عائلة **mega AVR** وبعض مُتحكمات عائلة **ATTiny**. وسيناقش هذا الفصل طريقة تشغيل **ADC** المدمج بعائلات AVR.

كيف يعمل المحول التناضري الرقمي

تحتوي معظم المُتحكمات الدقيقة (من مختلف الشركات) ومنها AVR على **ADC** من نوع **successive approximation adc** هذا المحول يعمل بطريقة مشابهه "الميزان القديم". تخيل معي أن لديك أحد هذه الموازين القديمة ذات الكفتين. ولنفترض أنك أردت أن تعرف وزن ثمرة بطيخ، فما الذي ستفعله؟

في البداية ستضع ثمرة البطيخ في أحد كفوف الميزان، ثم تدريجياً تبدأ باضافة وزن معروف مسبقاً في الكفة الأخرى فمثلاً قد تضع ربع كيلو جرام (250 جرام) ثم تتظر إلى الميزان وتقارن ارتفاع الكفتين، إذا لم تجد أن الكفتين قد تساواً ستضيف ربع كيلو جرام أخرى وهكذا .. يتم التوقف عن وضع المزيد من الوزن عندما تقترب كفتى الميزان من بعضها.

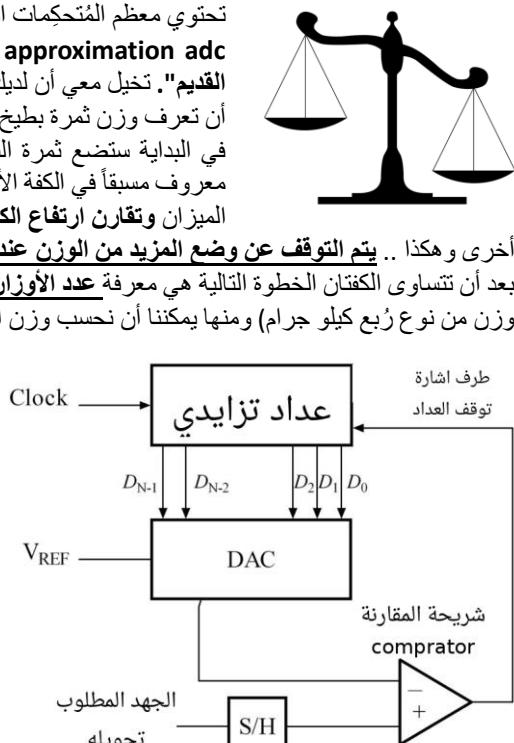
بعد أن تتساوى الكفتان الخطوة التالية هي معرفة عدد الأوزان التي تم وضعها في كفة القياس (مثلاً قد نجد أنه هناك 3 قطع وزن من نوع ربع كيلو جرام) ومنها يمكننا أن نحسب وزن الثمرة والذي يساوي $3 \times \frac{1}{4} \text{ كيلو جرام} = 750 \text{ جرام}$.

الخطوات السابقة تمثل نفس طريقة عمل **ADC**. حيث نجد أن **ADC** يحتوي على مجموعة مكونات تعمل مثل الميزان.

المكون الأول يسمى دائرة التقطيع (أخذ العينة) و تسمى **sample & hold (S/H) circuit**. هذه الدائرة تأخذ عينة من فرق الجهد المطلوب تحويله الصورة الرقمية وتدخلها للجزء التالي من **ADC**.

المكون الثاني يسمى بـ **analog comparator** وهو شريحة إلكترونية تماثل الميزان بالضبط و تمتلك طرفان للمقارنة. حيث تقوم بالمقارنة بين فرق جهد مطبق على طرفيها الأول (والذي يتم الحصول عليه من دائرة **S/H**) وفرق جهد مطبق على طرفيها الثاني.

المكون الثالث هو عداد رقمي تزايدى + محول رقمي - تماثلي **DAC**. هذان المكونان يعملان مثل الوزن المعروف مسبقاً. حيث نجد أن كلا المكونين يقومان بتوسيع جهد صغير وارساله إلى الطرف الثاني لشريحة **ADC**. وإذا لم يتساوى كلا الجهدتين يقوم العداد و **DAC** بزيادة الجهد مرة أخرى وهكذا.. وتنظر هذه العملية إلى أن تقوم شريحة **ADC** comparator بتساوي.



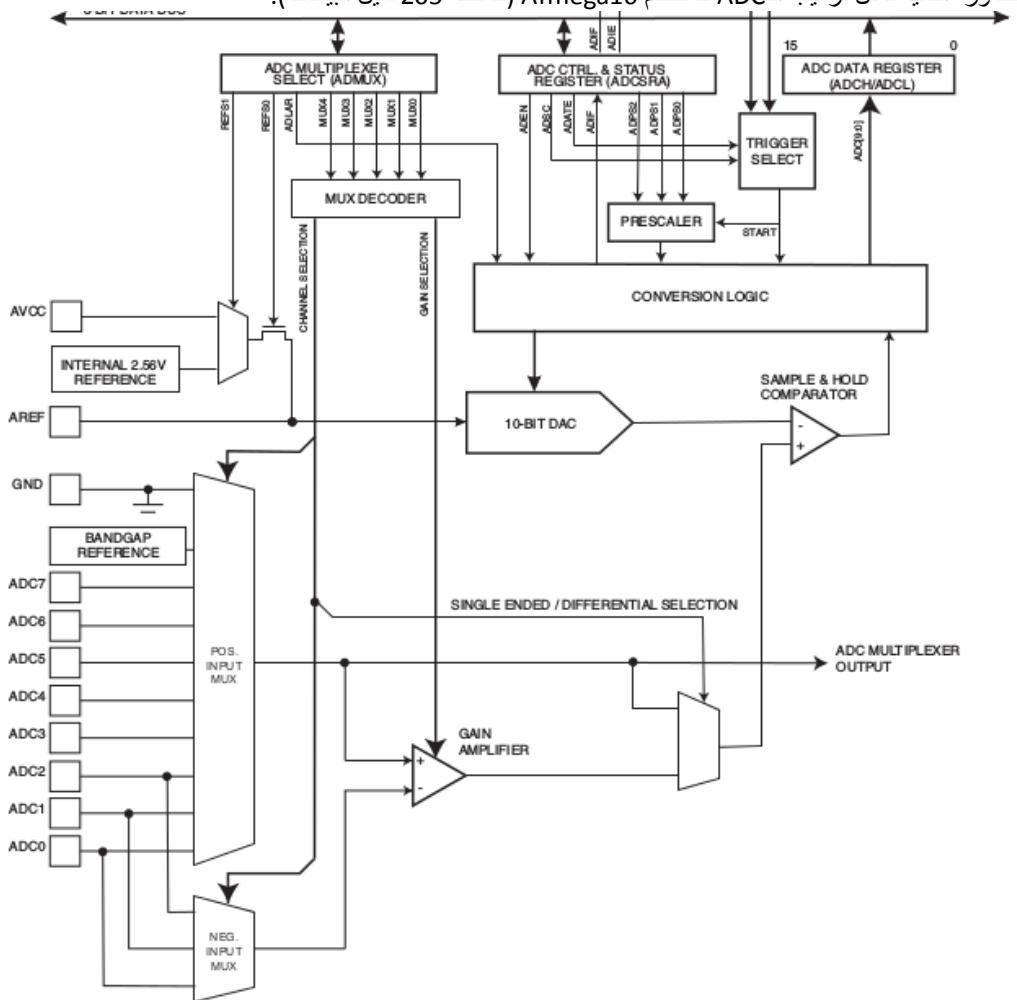


بالتبليغ أن الجهد المطبق من العداد والـ DAC قد تساوى أو تفوق على الجهد المطبق من عينة القياس.

عندما يحدث هذا التبليغ يتوقف العداد والـ DAC عن العمل ويتم تسجيل الرقم الذي توقف عنده. ويتم حفظ هذا الرقم في مسجلات خاصة تسمى ADCL و ADCH ومن خلال هذا الرقم يمكننا معرفة قيمة الجهد الذي تم ادخاله للـ ADC.

9.2 تركيب الـ ADC داخل المتحكم ATmega16

الصورة التالية تمثل تركيب الـ ADC للمتحكم ATmega16 (صفحة 205 دليل البيانات).



يتميز الـ ADC الموجود داخل ATmega16 بعده امكانيات مفيدة.

أولاً: يمكن تشغيل هذا الـ ADC في وضعين. وهما 8 bit و 10 bit الاختلاف الاساسي بين الوضعين هو "حساسية القياس" للجهد التنااري وأقل قيمة جهد يمكن قياسها. سيتم تناول الدقة 8 بت فقط في هذا الفصل.



ثانياً: يتصل دخل الـ ADC بشريحة analog multiplexer والتي تتصل أطرافها بجميع أطراف البورت A. هذه الشريحة تعمل كبوابة توصيل حيث يمكن ضبطها لتوصيل أي طرف في البورت A ليعمل كدخل للـ ADC وهذا يجعل التحكم يمتلك 7 أطراف كاملة يمكن استعمالها كدخل تماثلي ويسمى كل طرف "قناة دخل" input channel. مع مراعاة أنه يمكن تشغيل طرف واحد فقط في نفس الوقت (كما سنرى في الشرح لاحقاً).

أيضاً يحتوي الـ ADC على شريحة op-amp تعمل كمكثف جهد (يمكن تشغيله بصورة اختيارية) ودائرة مقارنة comparator تعمل على الأطراف الثلاثة الأولى PA0, PA1, PA2. (لن يتناول الكتاب شرح هذا الجزء ويمكنك الرجوع لدليل البيانات بدأً من الصفحة 205).

المُسجّلات

يحتوي الـ ADC على مجموعة من المُسجّلات سنسخدم 3 منهم لتشغيل وضع 8 بت. **ADMUX**: هذا المُسجل مسؤول عن اختيار الطرف الذي سيتم توصيله بالـ ADC لقياس الجهد كما يحتوي على مجموعة من البناء الهامة لضبط الـ analog Reference (انظر لشرح المثال الأول).

ADCSRA: المُسجل المسؤول عن تشغيل وإيقاف الـ ADC وكذلك التحكم في سرعة تشغيله. **ADCL & ADCH**: المُسجّلات المستخدمة في حفظ قيم القياس.

خطوات تشغيل الـ ADC

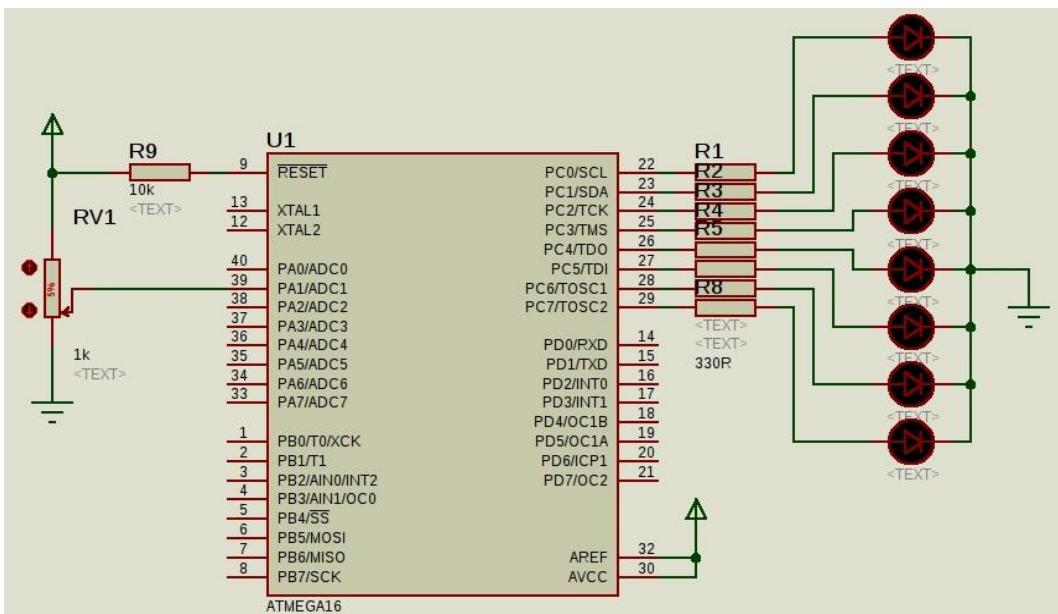
لقراءة الجهد التناهري يجب أن نقوم بمجموعة من الخطوات لتفعيل الـ ADC وضبطه بصورة صحيحة. هذه الخطوات هي كالتالي:

1. تفعيل الـ ADC (الوضع الافتراضي للـ ADC أنه غير فعال لذا سنقوم بتفعيله من المُسجل ADCSRA).
2. ضبط الـ Clock والتي تتحكم في سرعة تشغيل العداد الداخلي للـ ADC والذي بدوره يتحكم في سرعة قراءة الجهد.
3. اختيار الـ channel المطلوبة (طرف القياس) وذلك من خلال ضبط الـ analog multiplexer.
4. بدء عملية القياس والتحويل ثم قراءة قيمة الجهد.

9.3 المثال الأول: قراءة جهد متغير باستخدام مقاومة متغيرة

في هذا المثال سنسخدم المقاومة المتغيرة potentimeter كمصدر متغير للجهد. تتوارد هذه المقاومة في برنامج بروتوك باسم POT-HG أو Active potentiometer ويتم توصيل الطرف العلوي لها بالـ VCC والطرف السفلي بالـ GND أما الطرف الأوسط فسيكون مصدر الجهد المتغير.

في هذا المثال سنقوم بتوصيل طرف الجهد المتغير بالطرف PA1. كما سنستخدم 8 دايدات ضوئية لعرض قراءة الـ ADC على البورت C كما هو موضح في الصورة التالية:



الكود

```

#define F_CPU 1000000UL
#include <avr/io.h>
//adc لحفظ قيمة متغير
volatile uint8_t adcValue;

int main(void)
{
    // ضبط اعدادات البورتات
    DDRA = 0x00;
    DDRC = 0xff;
    // adc تشغيل ال
    ADCSRA |= (1 << ADEN);

    // اختيار معامل القسمة للد
    ADCSRA |= (1 << ADPS0) | (1 << ADPS1);

    // تفعيل وضع الـ8 بت
    ADMUX |= (1 << ADLAR);

    // اختيار القناة (الطرف) الذي سيتم قراءة الجهد المتغير منه
    ADMUX |= (1 << MUX0);

    while(1)
    {

```



```

// ابدء عملية التحويل
ADCSRA |= (1 << ADSC);

// انتظر حتى يتم الانتهاء من تحويل الجهد
while(ADCSRA & (1<<ADSC));

// ضع قيمة التحويل داخل المتغير
adcValue = ADCH;

// قم بعرض القيمة على البورت C
PORTC = adcValue;

}

return 0;
}

```

شرح الكود

في بداية البرنامج قمنا بعمل متغير 8 بت من نوع `uint8_t` باسم `adcValue`. س يستخدم هذا المتغير في حفظ القيمة الرقمية الناتجة من تحويل الجهد في الـ ADC. لكن هناك كلمة غريبة ظهرت لأول مرة بجانب المتغير وهي volatile فما هي؟

قبل أن نتعرف على هذه الكلمة. عليك أن تتعزز على أحد الخواص الهامة للمترجمات وتدعى **Code optimization**. كما نذكر من الفصول السابقة إن جميع المترجمات مهمتها هي تحويل اللغات عالية المستوى (مثل السي) إلى لغة التجميع Assembly والحقيقة أن الأمر الواحد من لغة السي قد يتحول إلى 1 أو 3 أو حتى 10 أوامر من لغة التجميع. المترجمات الذكية تستطيع اختصار عدد أوامر لغة التجميع وذلك لزيادة سرعة الكود (أوامر أقل = سرعة أكبر).

هنا يأتي دور الـ `code optimization`. وهي خاصية مدمجة في معظم المترجمات سواء المجانية مثل `gcc` أو المدفوعة مثل `IAR workbench`. وتهدف إلى اختصار أكبر قدر ممكן من أوامر الأSEMBLY. والآن نعود للبرنامج السابق.

سنجد أن المتغير `adcValue` يستخدم لنسخ قيمة المسجل `ADCH` (وهو المسجل الذي يحفظ فيه القيمة الرقمية لعملية التحويل). ثم نجد أن المتغير يستخدم في نقل هذه القيمة إلى البورت C في مجموعة الأوامر التالية:

```

// ضع قيمة التحويل داخل المتغير
adcValue = ADCH;
// قم بعرض القيمة على البورت C
PORTC = adcValue;

```

في الحالة الطبيعية إذا لم نكتب كلمة `volatile` سنجد أن المترجم قرر حذف المتغير `adcValue` لأنه بلا فائدة. وذلك لأنه من الممكن أن نستبدل كلا الأمرتين السابقتين بأمر واحد فقط وهو وضع قيمة `ADCH` مباشرة داخل `PORTC` دون الحاجة لنسخها لمتغير `adcValue`.

بالتأكيد يعد اختصار عدد الأوامر أمر مفيد جداً وسيجعل الكود يعمل أسرع. لكن في هذه الحالة قد يكون كارثي. لأننا قد نحتاج هذا المتغير في إجراء عمليات حسابية أخرى (كما سنرى في المثال القادم). وإذا قام المترجم بحذفه سينتسب ذلك الأمر في أخطاء غير معروفة. وهنا يأتي دور كلمة `volatile`. والتي تخبر المترجم أن يترك المتغير `adcValue` ولا يقوم بحذفه.

ملاحظة: اعتبرها قاعدة عامة، إذا كان هناك أي متغير أنت متأكد من تغير قيمته أثناء عمل المتحكم الدقيق ولا ترید للمترجم أن يحذفه فيجب أن تكتب كلمة `volatile` قبل المتغير أثناء تعریفه.



بعد الانتهاء من تعریف المتغير. نأتي للدالة الرئیسیه main حيث بدئنا بضبط الیورتات لیعمل الیورت A كدخل ویعمل الیورت C كخرج (النوصیل الدایودات الضوئیة الثمانیة).

DDRA = 0x00;

DDRC = 0xff;

ثم تبدأ مرحلة ضبط الـ ADC عبر سلسلة من الأوامر. في البداية يجب أن تشغّل الـ ADC لأنّ الوضع الافتراضي لـ ADC داخل جميع مُتحکمات AVR أنه غير مُفعّل لتوفیر الطاقة. ويتم ذلك عبر تغيير قيمة البت **ADEN** وهي اختصار ADC enable. تتوارد هذه البت داخل **ADCSRA**.

ADCSRA |= (1 << ADEN);

ADC Control and Status Register A – ADCSRA	Bit	7	6	5	4	3	2	1	0	ADCSRA
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0		
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		
Initial Value	0	0	0	0	0	0	0	0	0	

المرحلة الثانية هي ضبط الـ **clock** الخاصة بالـ ADC. فكما تعریفنا في بداية الفصل. يحتوي الـ ADC على عدد تزایدی يُستخدم لزيادة جهد المقارنة (الشبيه بالوزن المعياري في المیزان). ويحتاج هذا العداد الرقمی لـ **Clock** لتشغیله. المشكلة لوحیدة لهذا العداد هو أنه يجب أن يعمل ببطیء مقارنة بسرعة المُتحکم. فمثلاً لا يمكن أن تدخل الـ **clock** الرئیسیة للمُتحکم مباشرة إلى العداد (لأنّها تكون 1 میغا أو أعلى). لذا تستخدم شریحة تسمی بالـ **Prescaler Register** وهي شریحة تقوم بقسمة الـ **clock** الرئیسی على رقم محدد (من مضاعفات الرقم 2 مثل 2 – 4 – 8 – 16 – 32 – 64 – 128). يسمی هذا الرقم بـ **معامل القسمة** **Division Factor** ويتم تحديد هذا الرقم من البت **ADPS0** و **ADPS1** و **ADPS2** الموجودتان داخل المُسجّل **ADCSRA** من خلال الجدول التالي

Table 85. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

بالتأكيد سيظهر سؤال هام. على أي أساس نختار معامل القسمة وما هو تأثیره؟

في البداية هناك قاعدة عامة لجميع الـ ADC الموجودة داخل عائلة atmega وهي أن الـ **Clock** الخاص بالـ ADC يجب أن لا تزيد عن 128 كیلوهertz (ويمکن أن تكون أقل من ذلك). فمثلاً إذا كانت الـ **clock** الرئیسیة = 1 میجاهرتز إذا يجب أن نختار معامل قسمة = 8 أو أكثر وذلك لأنّ حاصل قسمة مليون $1 / 8 = 125$ ألف هertz وهو رقم أقل 128 کیلو.

لتأخذ مثال آخر. لنفترض أن سرعة الـ **clock** الرئیسیة كانت 4 میجاهرتز. فما هو معامل القسمة المناسب؟ سنجد أن المعامل المناسب هو 32 (أو أكثر) وذلك لأنّ حاصل قسمة 4 مليون $4 / 32 = 125$ ألف هertz. أو يمكن استخدام معامل قسمة 64 أو 126 فكلاهما سيجعل الـ ADC clock أقل بكثير من 128 کیلوهertz.

إذا ما الإختلاف بين أن اختار 32 أو 64 أو 128 إذا كانت كل هذه المعاملات مناسبة؟

الإختلاف الأساسي هو سرعة تحويل الجهد إلى قيمة رقمیة. فكلما زادت سرعة الـ ADC clock كلما استطاع أن يحول



الجهد إلى قيمة رقمية في زمن أقل.

لعد مرة أخرى للمثال السابق حيث كانت سرعة المتحكم الرئيسية 1 ميجا لذا إستخدمنا معامل قسمة = 8 وذلك عبر وضع قيمة 1 داخل كل من البت ADPS0 و ADPS1

ADCSRA |= (1 << ADPS0) | (1 << ADPS1);

الخطوة التالية هي اختيار وضع تشغيل الـ ADC (8 أو 10 بت). في حالة الـ 8 بت يتم وضع الرقم 1 داخل البت ADLAR الموجود في المُسجَل ADMUX. الوضع الإفتراضي لهذه البت = صفر (أي الـ ADC يعمل في وضع 10 بت).

// تعيين وضع الـ 8 بت

ADMUX |= (1 << ADLAR);

ADC Multiplexer Selection Register – ADMUX									
Bit	7	6	5	4	3	2	1	0	
Read/Write	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Initial Value	0	0	0	0	0	0	0	0	

عندما يعمل الـ ADC في وضع الـ 8 بت فإن قيمة تحويل الجهد يتم تسجيلها في المُسجَل ADCH وعندما يعمل في وضع الـ 10 بت فإن قيمة التحويل يتم تسجيلها في مسجلين (لأنها 10 بت ولا يكفي مُسجَل واحد لحفظ قيمتها) وهم ADCH و ADCL ويسمى هذا الوضع بالـ left adjusted حيث توضع أول 8 بتات من النتيجة في ADCL ويوضع آخر 2 بت في ADCH

وأخيراً نقوم بضبط الطرف الذي سيكون Input channel (الطرف الذي سنقيس من خلاله الجهد الكهربائي المتغير). في المثال السابق استخدمنا الطرف PA1 لهذا وجب أن يتم وضع 1 داخل البت MUX0

// اختيار القناة (الطرف) الذي سيتم قراءة الجهد المتغير منه

ADMUX |= (1 << MUX0);

حيث تتحكم هذه البت مع البتات (1,2,3,4)MUX في اختيار قناة الدخل كما هو موضح في الجدول التالي.

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input
00000	ADC0		
00001	ADC1		
00010	ADC2		
00011	ADC3	N/A	
00100	ADC4		
00101	ADC5		
00110	ADC6		

ملاحظة: الطرف ADC0 يتصل بالطرف PA0 والطرف ADC1 يتصل بالطرف PA1 وهكذا إلى نهاية الپورت A

بعد الانتهاء من ضبط الـ ADC يمكننا الآن أن نبدء عملية التحويل وقراءة الجهد. وسيتم ذلك عبر اعطاء الـ ADC الأمر ببدء عملية التحويل وذلك عبر وضع 1 داخل البت ADSC وهي اختصار لكلمة ADC start conversion

// ابدء عملية التحويل

ADCSRA |= (1 << ADSC);



هنا سينبأنا ADC بقراءة الجهد وتحويله إلى قيمة رقمية. ولكن كما نعرف مسبقاً أن ADC يعبر بطيء جداً مقارنة بالمحكم الدقيق لذا لا يمكننا أن نحصل على النتيجة فوراً ويجب أن ننتظر حتى ينتهي الـ ADC من عملية التحويل. ويتم ذلك عبر الأمر

انتظر حتى يتم الانتهاء من تحويل الجهد //

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

عند بدء عملية التحويل تظل البت ADSC قيمتها = 1 ولا تتحول إلى صفر إلا عند انتهاء عملية التحويل بنجاح. لذا استخدمنا الأمر السابق والذي يعني انتظار حتى تصبح البت ADSC قيمتها 0 وبذلك توقف الـ while.

وأخيراً يمكننا قراءة القيمة الرقمية من المسجل ADCH حيث نستطيع اما أن نستخدمها مباشرة أو ننسخها إلى أحد المتغيرات (في المثال السابق استخدمنا الورت C لعرض هذه القيمة وكذلك المتغير adcValue لننسخ هذه القيمة).

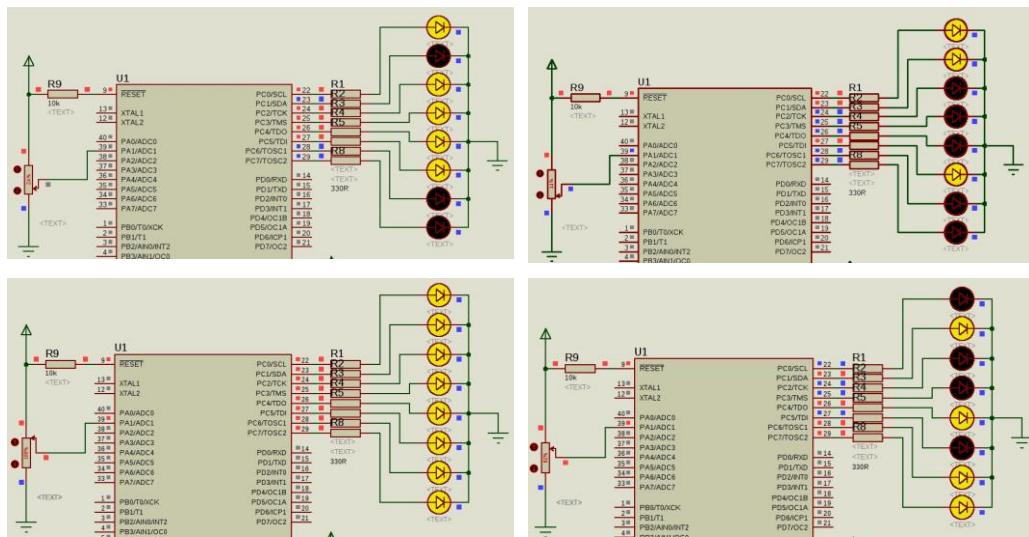
ضع قيمة التحويل داخل المتغير

adcValue = ADCH;

قم بعرض القيمة على الورت C

PORTC = adcValue;

قم بترجمة الكود السابق ثم ابدأ محاكاة المشروع على برنامج بروتس. يمكنك أن تحرك المقاومة المتغيرة لأعلى ولأسفل لتشاهد ان الدايوهات التمانية تعرض أرقام من 00000000 إلى 11111111 كما هو موضح في الصورة التالية.



9.4 حسابات الـ ADC

في المثال السابق استطعنا أن نحول الجهد التناضري المتغير إلى أرقام بين 00000000 إلى 11111111 (ما بين 0 إلى 255 بالصيغة العشرية). لكن هذه الأرقام لا تمثل قيمة فرق الجهد بصورة مباشرة لذا سنتعرف على في هذا الجزء على بعض المعادلات البسيطة التي ستساعدنا على حساب فرق الجهد وكذلك حساب قيم الحسات التمانية.

تحويل القيمة الرقمية إلى فرق جهد

المعادلة التالية تقوم بتحويل القيمة الرقمية إلى فرق جهد.



$$Voltage = \frac{DigitalValue * Vref}{2^n}$$

- Digital Value: القيمة الرقمية الموجودة في ADC (بالصيغة العشرية)
- Vref: الجهد المرجعي للـ ADC (أنظر للشرح بالأسفل)
- n: وضع دقة التشغيل 8 بت أو 10 بت

الـ Vref هو الجهد المرجعي الذي يعتمد عليه الـ ADC لحساب الجهد المترافق. هذا الجهد يساهم في تحديد حساسية قياس الـ ADC (كما سنرى في الجزء التالي) ويتم تحديده من خلال البناة REF0 و REF1. في المثال السابق تركنا هذه البناة على الوضع الافتراضي (صفر) والذي يعني أن الجهد المرجعي $Vref = Vcc = 5 \text{ volt}$.

الصورة التالية توضح طريقة تغيير الـ Vref بالتغيير قيمة REF0 ، REF1 في المُسجِّل ADMUX

• Bit 7:6 – REF0:1: Reference Selection Bits

These bits select the voltage reference for the ADC, as shown in **Table 83**. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

Table 83. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

لنقترض أن القيمة الرقمية التي حصلنا عليها = 128 وكان الـ ADC يعمل على الوضع 8 بت. إذا بالتعويض في المعادلة السابقة نجد أن قيمة الجهد الذي تم قياسه يساوي $128 * \frac{5}{256} = \frac{128 * 5}{256} = 2.5 \text{ volts}$ $\frac{56 * 5}{256} = \frac{56 * 5}{256} = 1.09 \text{ volts}$ = 1.09 volts. إذا بالتعويض في المعادلة السابقة نجد أن قيمة فرق الجهد هي = 0.56.

حساسية القياس

تعرف حساسية القياس بأنها أقل جهد يمكن للـ ADC أن يقيسه ويُعرف عليه بصورة صحيحة. وتعتمد حساسية القياس للـ ADC على عاملين وهما وضع التشغيل (8 أو 10 بت) والجهد المرجعي Vref. كلما العاملين يؤثران بشكل كبير جداً في دقة القياس. المعادلة التالية تمثل حساسية القياس.

$$sensitivity = \frac{Vref}{2^n}$$

مجدداً تمثل n وضع القياس (8 أو 10 بت). فمثلاً لو أن $Vref = Vcc = 5 \text{ volt}$ ويُعمل الـ $Vref = Vcc = 5 \text{ volt}$ بدقة 8 بت فهذا يعني أن حساسية القياس تساوي

$$\frac{5}{2^8} = \frac{5}{256} = 0.019 \text{ volt}$$

مما يعني أن أقل جهد يمكن قياسه هو 0.019 فولت (نحو 20 مللي فولت).

يتحكم الـ Vref في أقصى جهد يمكن قياسه فمثلاً لو كان $Vref = 3 \text{ volt}$ فولت إذا أقصى جهد يمكن للـ ADC أن يقيسه هو 3 فولت



صفحة جديدة.....

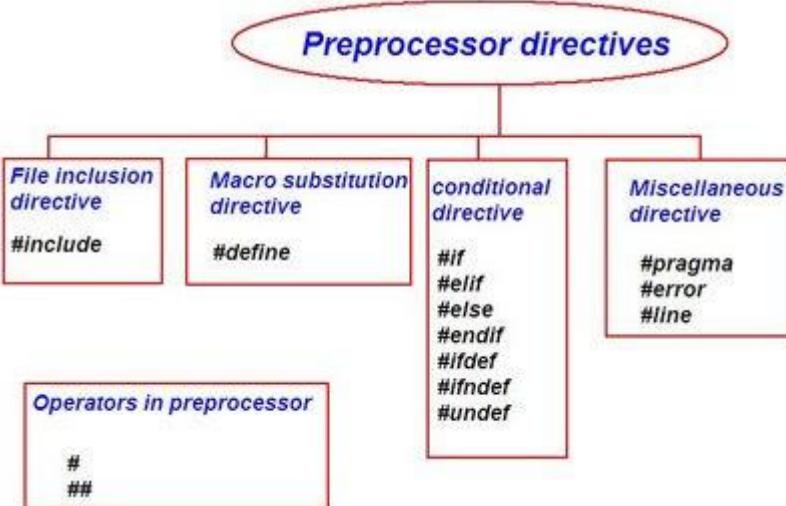


10. المعالج التمهيدي وصناعة المكتبات البرمجية

الفصل
عن أكواد

في هذا
سنتحدث
عن

C



حيث سنعرف على الفارق بين الأوامر التنفيذية والأوامر التوجيهية وأهميتها بصورة مفصلة مثل الأمر `#include` وكذلك `#define` وكذلك سنعرف على كيفية صناعة المكتبات البرمجية `libraries`. مع شرح مثال لعمل `uart driver` على صورة مكتبة.

- ✓ الأوامر التنفيذية والأوامر التوجيهية
- ✓ بعض استخدامات `#` Preprocessor
- ✓ قواعد صياغة الأوامر التوجيهية
- ✓ طرق كتابة `Function-like macros`
- ✓ تصميم المكتبات البرمجية
- ✓ مثال: تصميم مكتبة `UART driver`



10.1 الأوامر التنفيذية والأوامر التوجيهية

الأوامر التنفيذية هي أي أمر مباشر في لغة السي مثل – if وجميع الأوامر الرياضية أو المنطقية – AND – OR – NOT – XOR، كل هذه الأوامر تستخدم "تنفيذ" أمر مطلوب. على العكس الأوامر التوجيهية Directive (تسمى أيضاً أوامر إرشادية) هي أوامر لا تدخل مباشرة في ترکيب الكود ولكن تستخدم في إرشاد المترجم compiler لأداء بعض الأمور.

المترجم GCC يعرف مسبقاً جميع أوامر لغة السي المعيارية لكنه لا يعرف الدالة delay_ms والتي لا تتوارد إلا في المكتبة البرمجية delay.h لذا تستخدم الأمر التوجيهي

```
#include <delay.h>
```

وذلك لنرشد المترجم للمكان الذي يحتوي على دوال التأخير الزمني التي سنحتاجها في البرامج وذلك حتى يستطيع المترجم أن يصل إلى هذه الدوال بصورة صحيحة ويقوم بتحويلها إلى صيغة الهيكس.

C - preprocessors بعض استخدامات

الأوامر التوجيهية ليست مقتصرة فقط على الأمر include وإنما هناك العديد من الأوامر منها:
أمر لتضمين ملف أو مكتبة معينة - **include**

أمر لتعريف دلالات الكلمات كأن نعرف أنه كلما وردت كلمة Pi في الكود فهو يعني 3.14 وكذلك لإعطاء معرف(رمز) لكتلة أسطر برمجية مع إمكانية وجود وسطاء arguments وهذا ما يسمى- function-.like macros

أمر لتحديد بعض الأوامر للمترجم compiler - **pragma**

الترجمة الشرطية conditional compilation - مجموعة أوامر تستخدم في الترجمة حسب شروط معينة،
مثال: نخبر المترجم أنه لو كنت في نظام تشغيل ويندوز فقم بتضمين المكتبة الفلانية أما لو كنت في نظام تشغيل لينكس فقم بتضمين مكتبة أخرى.

C - preprocessors syntax قواعد الأوامر التوجيهية

أي سطر برمجي يبدأ برمز المربع # فإن ما يليه هو أمر سيوجه إلى ال preprocessor .

#include

يعتبر الأمر أكثر الاستخدامات شيوعاً من أوامر -preprocessor C وهو أمر توجيهي من أجل تضمين مكتبة (وهي عبارة عن مجموعة تعاريفات) أو حتى ملفات مصدرية أخرى.

مثال:

بفرض الكود المصدري لملف main.c هو التالي:

```
#include <avr/delay.h>
#include "device_conf.h"
```

```
int main(void)
{
```



```

set_port_output;
while(1)
{
    PORT_on;
    _delay_ms(1000);
    PORT_off;
    _delay_ms(1000);
}
return 0;
}

```

نلاحظ أن التضمين الأول كان بين قوسين <> والثاني بين إشارتين ""، حيث أن الاستخدام الأول يكون عندما نريد أن يتم البحث عن الملف المحدد ضمن المسارات الممتاحة في إعدادات المترجم. والاستخدام الثاني يكون عندما نريد البحث عن الملف المحدد ضمن المجلد المصدري نفسه.

أيضاً نلاحظ وجود بعض الأوامر الغريبة مثل `PORT_on` و `PORT_of`. الحقيقة أن تقسير هذه الأوامر يقع في الملف `device_conf` والذي يحتوي على الكود التالي:

```

#include <avr/io.h>
#define PORT_on PORTD=0xFF
#define PORT_off PORTD=0x00
#define set_port_output DDRD=0xFF

```

عند القيام بعملية الترجمة **Compiling** ما سيحدث أن المعالج التمهيدي للغة السي سيبدأ معالجة الكود قبل عملية الترجمة الحقيقة (تحويل الملف إلى هيكس) وسيتم معالجة الأمر `include` ليصبح البرنامج السابق كالتالي:

(لتبسيط لن نذكر نتائج تضمين مكتبة `avr/delay.h` أو مكتبة `avr/io.h`)

```

#define PORT_on PORTD=0xFF
#define PORT_off PORTD=0x00
#define set_port_output DDRD=0xFF

int main(void)
{
    set_port_output;
    while(1)
    {
        PORT_on;
        _delay_ms(1000);
        PORT_off;
        _delay_ms(1000);
    }
}

```

نتيجة تنفيذ الأمر
include



```

    }
return 0;
}

```

#define

يُستخدم هذا الأمر التوجيبي directive لتعريف كلمات كرموز (كلمات مفاتيحية) ترد في الكود ليتم استبدالها بالقيمة المحددة (أرقام أو حروف أو أسطر برمجية أخرى) مع أننا سنرى لاحقاً أن ال preprocessor لا يستطيع التمييز بينها.

ففي الكود السابق سيقوم ال preprocessor بالبحث عن أماكن ورود PORT_on و PORT_off و set_port_output بالبحث عن PORTD=0xFF و PORTD=0x00 و يقوم باستبدالها بما هو منكور في تعريفها. أي أن الكود سيصبح كالتالي (بعد أن ينتهي المعالج من معالجة الأمر):

```

#define PORT_on PORTD=0xFF
#define PORT_off PORTD=0x00
#define set_port_output DDRD=0xFF

```

```

int main(void)
{
set_port_output;
while(1)
{
    PORTD=0xFF;
    _delay_ms(1000);
    PORTD=0x00;
    _delay_ms(1000);
}
return 0;
}

```

نتيجة تنفيذ الأمر
define

ملاحظة مهمة: إن ما يرد بعد الأمر التوجيبي لا يتم معالجته أو تنفيذه أخطائه، مثلاً لو قمنا بالتعديل التالي وهو إضافة تعليق

```

#define PORT_on PORTD=0xFF
#define PORT_off PORTD=0x00      //set portD off
#define set_port_output DDRD=0xFF

```

سيصبح الكود الرئيسي بعد ال Preprocessor كالتالي:

```

while(1)
{
    PORTD=0xFF;
    _delay_ms(1000);
    PORTD=0x00      //set portD off ;
    _delay_ms(1000);
}

```



نلاحظ أن الكود أصبح يحتوي على خطأ حيث أصبحت الفاصلة المنقطة بعد التعليق، فالـ **preprocessor** ليس من مهمته فهم ما بعد الأوامر التوجيهية وإنما فقط استبدال الرموز بقيمها.

function-like macros 10.3

يمكننا من خلال الـ **C preprocessor** إنجاز ما يشبه **الدواال** ولذلك تسمى **function-like macros** مع اختلاف جوهري بين الدواال و **function-like macros**، شكل الشبه الوحيد هو إمكانية إنجاز **macros** يمكنه أن يستقبل بعض المتغيرات، أما من الناحية التنفيذية فليس هناك أي وجه شبه:

- **function** – كتلة من الكود البرمجي يمكن أن تستدعيها بأي مكان من البرنامج الأساسي لتقوم بالتنفيذ واستخدام المتغيرات في حال وجودها مع إمكانية ارجاع قيمة بعد التنفيذ.
- **Function-like macros** – مجموعة من الأسطر البرمجية المختصرة برمز، و عند إبراد هذا الرمز في الكود فهو بمثابة إبراد هذه الأسطر البرمجية كما هي، بخلاف مبدأ الدواال المعتمد على الاستدعاء مع وجود الكود دون تكرار.

لابد من التوجيه إلى الجانب السلبي لاستخدام الـ **C preprocessor** لإنجاز ما يشبه الدواال **function-like macros** يؤدي إلى تضاعف حجم البرنامج، وذلك لأن استخدام الـ **C preprocessor** لا يتعدي في النهاية عن اختصار أسطر برمجية بكلمات مقتاحية.

macros syntax 10.4

لصناعة **macro** يمكنه استقبال متغيرات يجب أن تتبع اسم المايكرو (الرمز) بأقواس تحوي أسماء المتغيرات مباشرة ، دون تحديد نوعها كما في الدواال، إذ لا يهم الـ **function-like macros** نوع المتغيرات التي يتم تمريرها إليها. المجموعة التالية من الأكواد تمثل أشهر الـ **macros** التي يتم استخدامها عادة لتسهيل برمجة المُتحكمات واختصار الكثير من الوقت. مع العلم أن هذه الـ **macros** يمكن استخدامها مع أنواع مختلفة من المُتحكمات الدقيقة عبر أي مترجم يدعم لغة السي المعيارية فيمكنك مثلاً أن تستخدمها في برمجة مُتحكمات ARM cortex.

```
#define BIT_SET(ADDRESS,BIT) (ADDRESS |= (1<<BIT))
#define BIT_CLEAR(ADDRESS,BIT) (ADDRESS &= ~(1<<BIT))
#define BIT_CHECK(ADDRESS,BIT) (ADDRESS & (1<<BIT))
#define BIT_FLIP(ADDRESS,BIT) (ADDRESS ^= (1<<BIT))
```

لأخذ أحد الـ **function-like macros** كمثال

```
#define BIT_SET(ADDRESS,BIT) (ADDRESS |= (1<<BIT))
```

اسم **BIT_SET** هو macro و المتغيرات التي يستقبلها هي ADDRESS, BIT حيث يمكننا أن نستخدمه داخل البرنامج الرئيسي كالتالي:

```
BIT_SET(PORTB,5);
```

وهذا مكافئ تماماً للأمر الذي يقوم بوضع 1 داخل أي بت من أي مُسجل مطلوب كالتالي:

```
PORTB |= (1<<5);
```

أما المايكرو **BIT_CLEAR** فهو مكافئ للأمر الذي يضع 0 داخل أي بت من أي مُسجل كالتالي:

```
PORTB &= ~(1<<5);
```

مراجع إضافية 10.5

سلسلة مقالات الـ **C** – العربية (المصدر الأصلي للجزء المشروح بالأعلى)
<http://www.atadiat.com/c-preprocessor-part1>



<http://www.atadiat.com/c-preprocessor-part-2>
<http://www.atadiat.com/c-preprocessor-part-3>

مراجع أجنبية إضافية

<http://www.mybitbox.com/2012/12/robust-c-code-part-3-wrapping-c/>
<http://www.mybitbox.com/2012/12/robust-c-code-part-2-advanced-c-preprocessor/>
<http://www.mybitbox.com/2012/11/robust-c-code-part-1-c-preprocessor/>
<http://www.cprogramming.com/tutorial/cpreprocessor.html>
<http://www2.hh.se/staff/vero/embeddedProgramming/lectures/printL2.pdf>
<http://www.phaedsys.com/principals/bytetechnology/bytetechnologydata/bcfirststeps.pdf>
http://en.wikipedia.org/wiki/C_preprocessor



10.6 تصميم المكتبات البرمجية في لغة السي

المكتبات البرمجية تعد من أفضل أساليب "تجزئة الكود" فهي تسمح للمبرمجين بصناعة نماذج من الأكواد البرمجية التي يمكن إعادة استخدامها بسهولة في أكثر من تطبيق. مثلاً لنفترض أنك تريدين تشغيل شاشة أو محرك أو حساس خاص مع المتحكم الدقيق، يمكنك أن تكتب الأكواد الخاصة بتشغيله (أو كما تسمى **driver code**) داخل البرنامج الرئيسي وينتهي الأمر عند هذا الحد. لكن عندما تريدين أن تستخدم نفس الحساس في مشروع آخر سيتوجب عليك أن تعيد كتابة هذا الكود مرة أخرى.

المكتبات البرمجية توفر الوقت والمجهود لإعادة كتابة هذه الأوامر فكل ما عليك فعله هو أن تصمم **الـ driver code** مرة واحدة وتضعها في **library** ثم تستخدم هذه المكتبة في أي مشروع ترغب به.

ملاحظة: تسمى المكتبات التي تشغّل أي جزء داخل المتحكم الدقيق مثل **GPIO, i2C, UART, ADC** أو أي مكون إلكتروني خارجي **Software driver**

أيضاً تساعد المكتبات على تسهيل العمل الجماعي بين الأفراد. فمن الصعب على مجموعة مكونه من 10 مبرمجين مثلاً أن يكتبوا كل الأكواد داخل ملف واحد فقط. وبدل من ذلك يتم تقسيم الأكواد الكبيرة إلى مجموعة من **Modules** (الوحدات) والتي عادة تكون مجموعة من المكتبات البرمجية ويتولى كل شخص من فريق التطوير العمل على أحد هذه الوحدات بالتزامن مع باقي الفريق.

تركيب المكتبات في لغة السي

ت تكون المكتبات في لغة السي عادة من بناء نوعين من الملفات، الأول يسمى **Header file** أو كما يحب البعض أن يسميه **Implementation file** (ملف التعريفات) ويكون على هيئة الامتداد **.file.h** والثاني هو الملف التطبيقي **.file.c** والذي يحتوي على الأوامر الحقيقة للمكتبة ويكون من نوع ملفات السي **.file.c**

مثلاً لنفترض أننا نريد تصميم مكتبة لتشغيل الاتصال التسلسلي **UART** وذلك لتسهيل استخدام **UART** دون الحاجة لإعادة كتابة جميع الدوال في كل برنامج.

10.7 خطوات صناعة المكتبة

تمر أي مكتبة برمجية بمجموعة من الخطوات حتى نحصل على كود فعال ويعمل بصورة جيدة. ويحسن أن تتبع هذه الخطوات في أي من المكتبات التي ستقوم بكتابتها مستقبلاً.

الخطوة الأولى: فهم الكود والتجربة الأولية

في البداية عليك أن تفهم العنصر أو الجهاز الذي تريدين تكتيب له هذا **الـ driver** وذلك عبر اختبار بعض الأمثلة في البرنامج الأساسي أو لا ثم تحويل هذه الأمثلة لمكتبة. بما أننا سنكتب **UART driver** فهذه الخطوة قد تمت بالفعل في الفصل الخاص بالـ **UART** حيث يمكنك الرجوع إليه مرة أخرى لمراجعة الأوامر والدوال التي سنتخدمها في الخطوات التالية.

الخطوة الثانية: تكوين ملف الهيدر **uart.h**

بعد القيام بجميع التجارب التي تختبر الأكواد التي سنحتاجها سنقوم بعمل ملف **الهيدر** **Header file** والذي يتكون من مجموعة من الأوامر **preprocessor** - **C** مضافة إليها جميع أسماء الثوابت والمتغيرات وكذلك **function prototypes** (أسماء الدوال). الكود التالي يمثل الهيكل الأساسي لأي ملف هيدر (سواء لغة السي أو السي **++**). ويمكنك عمل هذا الملف باستخدام أي محرر نصوص مثل **notepad++**

```
#ifndef __LIBRARYNAME_H__
#define __LIBRARYNAME_H__
```



هنا تكتب جميع التعريفات للدوال والمتغيرات المختلفة

#endif

يتم استبدال LIBRARYNAME باسم المكتبة المطلوب مع مراعاة أن تكون جميع الأحرف من نوع upper-case (حروف كابيتال) فمثلاً لو كان اسم المكتبة **uart** سنكتب داخل ملف الهيدر الصيغة التالية (أيضاً لا تنسى كتابة **_H** بعد اسم المكتبة):

```
#ifndef __UART_H__  
#define __UART_H__
```

.....

#endif

لأخذ مثال آخر. لنفترض أن اسم المكتبة هو **uartDriver** فسيكون شكل ملف الهيدر كالتالي:

```
#ifndef __UARTDRIVER_H__  
#define __UARTDRIVER_H__
```

.....

#endif

الآن يمكننا أن نكتب جميع التعريفات المطلوبة والتي تشمل أسماء المتغيرات والثوابت وكذلك أسماء الدوال التي سنستخدمها وأي C – preprocessor قد تحتاجها لتشغيل أوامر المكتبة. إذا كنت تذكر من الفصل الخاص بالـ UART ستجد أنها هناك مجموعة متغيرات ودوال أسماؤها كالتالي:

المتغيرات

uint16_t UBRR_Value // المتغير المسؤول عن تحديد سرعة السيريرال
الدوال

void UART_init() // تشغيل وضبط

void UART_send_char(char data) // إرسال حرف

char UART_receive_char() // استقبال حرف

void UART_send_string(char *data) // إرسال كلمة أو جملة نصية

الآن كل ما عليك فعله هو إضافة كل التعريفات السابقة إلى ملف الهيدر مع إضافة علامة الفاصلة المنقوطة **(;)** بعد كل تعريف (سواء كان متغير أو دالة). ليصبح شكل ملف الهيدر:

```
#ifndef __UARTDRIVER_H__  
#define __UARTDRIVER_H__  
  
uint16_t UBRR_Value;  
void UART_init(uint16_t baud_rate);  
void UART_send_char(char data);  
char UART_receive_char();  
void UART_send_string(char *data);
```



#endif

تتبقى خطوة أخيرة للانتهاء من الملف وهي إضافة المكتبات التي قد تحتاجها الدوال السابقة. إذا كنت تذكر الدالة `UART_init`() كانت تستخدم المكتبة البرمجية `math.h` لحساب الرقم المسؤول عن تحديد سرعة الـ `UART` وكذلك تستخدم المكتبة `avr/io.h` في ضبط قيم المسجلات لذا سنضيف كلا المكتبيتين إلى ملف الهيدر. أيضاً نحتاج أن نكتب تعريف سرعة المُتحكم لأننا سنحتاج `UART_init` داخل الدالة `F_CPU` كما يُحسن أن نضيف بعض التعليقات في بداية المكتبة لتوضح تاريخ صناعة هذه المكتبة والوظائف التي تحققها. الشكل النهائي يمثل الصورة النهائية من ملف الهيدر.

```
/*
 *      UART driver v.0.1
 *      Date: 9-2015
 * This library designed to make a simple and re useable UART driver
 * for both ATmega16 and ATmega32
 */

#ifndef __UARTDRIVER_H__
#define __UARTDRIVER_H__

#define F_CPU 16000000UL
#include <avr/io.h>
#include <math.h>

uint16_t UBRR_Value; // Variable to store the calculations needed to
set speed

void UART_init(uint16_t baud_rate); // Initiate the UART
void UART_send_char(char data); // Send single character
char UART_receive_char(); // Receive single character
void UART_send_string(char *data); // Send array of characters

#endif
```

ملاحظة: الملف `uart.h` و `uart.c` مرفقان مع الكتاب لتسهيل نسخ النصوص أو تعديلها

الخطوة الثالثة: تكوين ملف التطبيق `uart.c`

الخطوة الأخيرة هي كتابة الأوامر الحقيقة لجميع الدوال داخل ملف جديد بنفس اسم المكتبة ولكن بامتداد `.c` مثل أن نكتب `uart.c` مع ملاحظة أنه لا يتم تعريف أي متغيرات جديدة داخل هذا الملف (لأننا قد كتبنا تعريفات المتغيرات في ملف الهيدر).

كما يجب أن يحتوي ملف التطبيق على أمر التضمين `include` وذلك لإضافة ملف الهيدر إلى ملف التطبيق كما هو موضح في الكود التالي (والذي يمثل محتوى الملف `uart.c`).

```
#include "uart.h"

void UART_init(uint16_t baud_rate)
```



```

{
    uint16_t UBRR_Value = lrint ( (F_CPU / (16L * baud_rate) ) -1);
    UBRRL = (uint8_t) UBRR_Value;
    UBRRH = (uint8_t) (UBRR_Value >> 8);
    UCSRB = (1<<RXEN) | (1<<TXEN);
    UCSRC |= (3<<UCSZ0);
}

void UART_send_char(char data)
{
    while( ! (UCSRA & (1<<UDRE) ) );
    UDR = data;
}

char UART_receive_char()
{
    while (! (UCSRA & (1 << RXC) ) );
    return UDR;
}

void UART_send_string(char *data)
{
    while(*data > 0)
        UART_send_char(*data++);
    UART_send_char('\0');
}

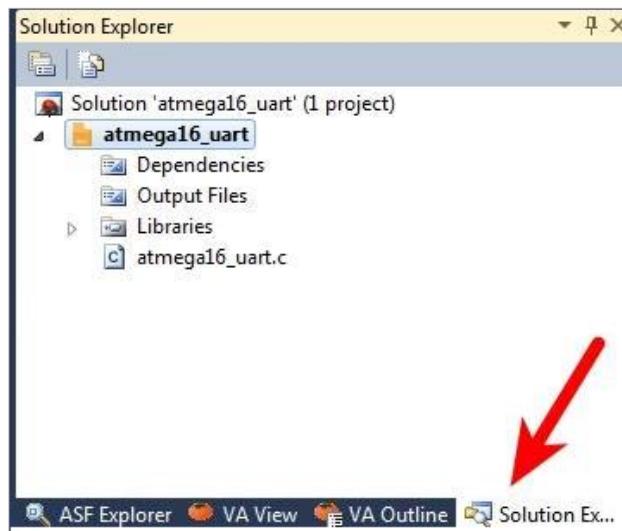
```

10.8 تجربة المكتبة في برنامج Atmel studio

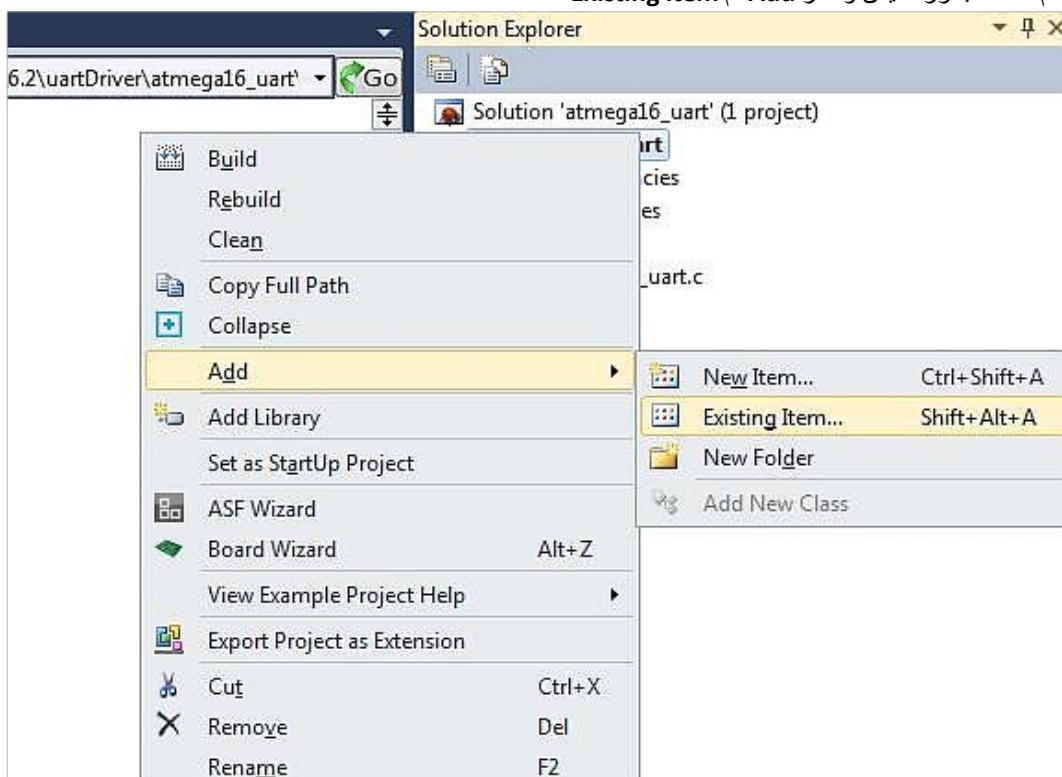
والآن بعد أن انتهينا من كتابة الـ **UART driver** لنقم باختباره. في البداية سنقوم بإضافة ملفات المكتبة إلى مشروع جديد داخل برنامج **Atmel studio** ويمكنك ذلك بطريقتين، الأولى أن تنسخ ملف **uart.c** و **uart.h** إلى نفس مجلد المشروع **C:/documents/atmel** أو أن تقوم بذلك بسهولة من متصفح ملفات المشروع من داخل البرنامج نفسه.



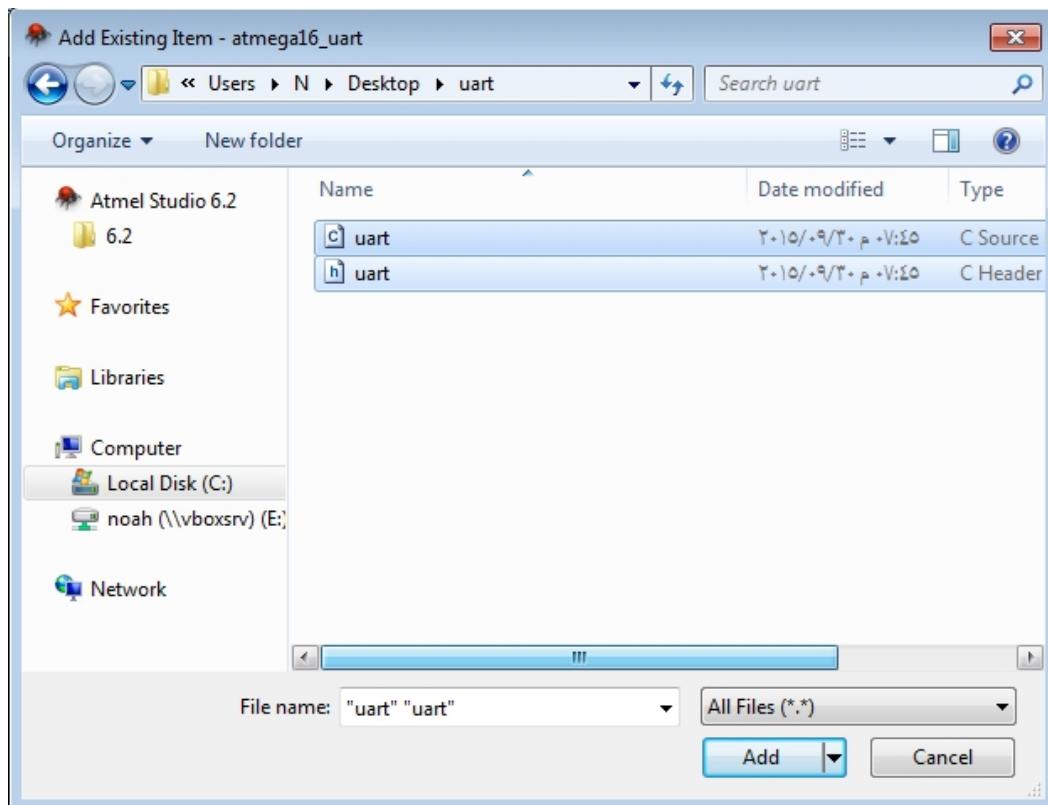
في البداية توجه إلى **Solution Explorer** من الجانب الأيمن من البرنامج



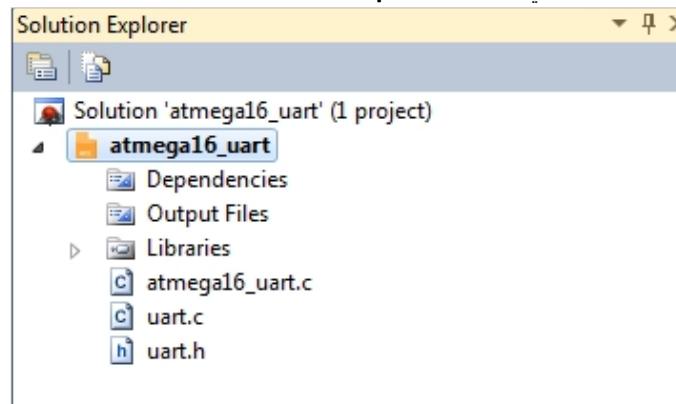
ثم اضغط بالزر الأيمن وأختر **Add** ثم **Existing item...**



والآن اختر الملفين **uart.h** و **uart.c**



بعد إضافة الملفات ستلاحظ ظهورها في قائمة



والآن لنكتب برنامج بسيط لإرسال حرف H باستخدام الـ **uart driver**. في البداية سنقوم بعمل **include** للملف **uart.h** داخل البرنامج الأساسي (لاحظ أنه لا داعي لإعادة كتابة **define F_CPU** ولا داعي لإضافة المكتبة **avr/io.h** لأن **.uart.h** كلاهما مضافة بالفعل داخل الملف **uart.h**).

```
#include "uart.h"
#include <util/delay.h>

int main(void)
```

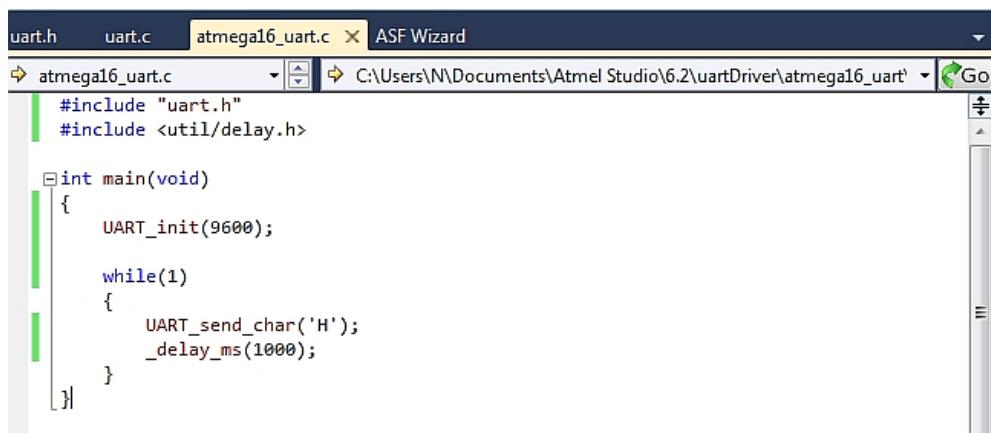


```
{
    UART_init(9600);

    while(1)
    {
        UART_send_char('H');
        _delay_ms(1000);
    }

return 0;
}
```

صورة الكود الذي سيختبر المكتبة داخل برنامج Atmel studio



لاحظ أن أمر تضمين المكتبة `uart.h` يكون مكتوب بين علامتي "" وذلك لأن ملف المكتبة يكون موجود داخل نفس مجلد المشروع بينما المكتبات الأخرى مثل `delay.h` تكتب بين علامتي <> لأنها تكون موجودة داخل مجلد المكتبات العام التابع لـ `.toolchain`

الصور التالية تمثل شكل الملف `uart.h` و `uart.c` داخل برنامج Atmel Studio



uart.h X uart.c atmega16_uart.c ASF Wizard

uart.h C:\Users\N\Documents\Atmel Studio\6.2\uartDriver\atmega16_uart\ Go

```
/*
 *      UART driver v.0.1
 *      Date: 9-2015
 *      This library designed to make a simple and re usable UART driver
 *      for both atmega16 and atmega32
 */

#ifndef __UARTDRIVER_H__
#define __UARTDRIVER_H__

#define F_CPU 16000000UL
#include <avr/io.h>
#include <math.h>

// Variable to store the calculations needed to set speed
uint16_t UBRR_Value;

void UART_init(uint16_t speed);      // Initiate the UART
void UART_send_char(char data);      // Send single character
char UART_receive_char();           // Receive single character
void UART_send_string(char *data);   // Send array of characters

#endif
```

الملف uart.c

uart.h X uart.c atmega16_uart.c ASF Wizard

UART_init.UBRR_Value uint16_t UBRR_Value = lrint((F_CPU / (16L * baud_rate)) -1) Go

```
#include "uart.h"

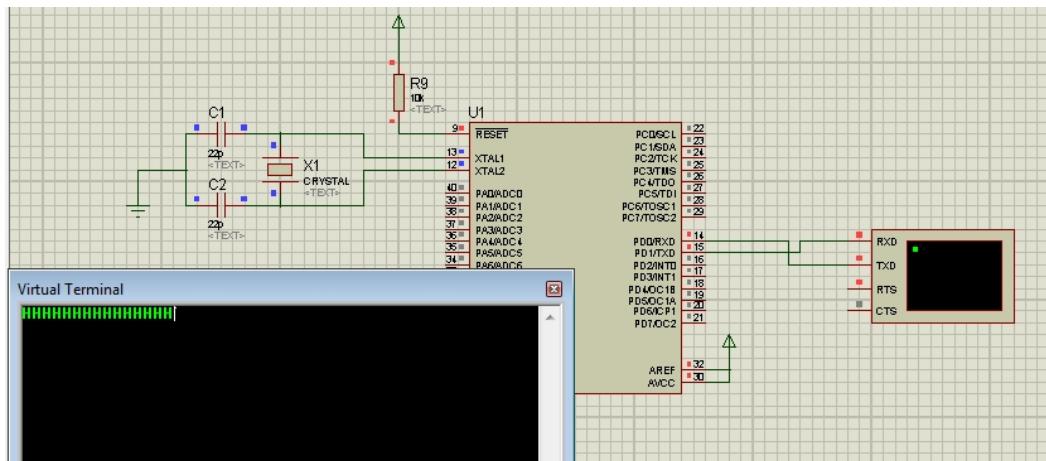
void UART_init(uint16_t baud_rate)
{
    uint16_t UBRR_Value = lrint( (F_CPU / (16L * baud_rate)) -1);
    UBRRL = (uint8_t) UBRR_Value;
    UBRRH = (uint8_t) (UBRR_Value >> 8);
    UCSR_B = (1<<RXEN) | (1<<TXEN);
    UCSR_C |= (3<<UCSZ0);
}

void UART_send_char(char data)
{
    while( ! (UCSRA & (1<<UDRE) ) );
    UDR = data;
}

char UART_receive_char()
{
    while ( ! (UCSRA & (1 << RXC) ) );
    return UDR;
}
```



بعد الانتهاء من كتابة الكود قم بترجمته واستخلاص ملف الهيكس ثم قم بمحاكاة البرنامج بنفس الدائرة المذكورة في المثل الأول في الفصل الخاص بالـ **UART**. والصورة التالية تمثل نتيجة المحاكاة.





صفحة جديدة.....



11. أنظمة الوقت الحقيقي RTOS



يشرح هذا الفصل استخدام أنظمة تشغيل الوقت الحقيقي Real Time OS لتشغيل المهام المتعددة Multitasking وأنظمة الاستجابة السريعة. حيث سيتم تناول نظام FreeRTOS في هذا الفصل باعتباره أفضل نظام RTOS مجاني (ومفتوح المصدر).

- ✓ مقدمة عن مفهوم الـ RTOS
- ✓ كيف يتم تنفيذ أكثر من مهمة
- ✓ كيف تعمل نواة النظام FreeRTOS Kernel
- ✓ تشغيل FreeRTOS على أي متحكم System Porting to any AVR
- ✓ إعدادات النظام
- ✓ مثال: تشغيل 3 مهامات مختلفة
- ✓ شرح طرق إدارة المهام



11.1 مقدمة عن أنظمة الوقت الحقيقي Real Time Systems



تعرف أنظمة الوقت الحقيقي بأنها أنظمة الحواسيب أو الأنظمة المدمجة التي تستخدم في أداء مجموعة من المهام تتطلب كل مهمة استجابة زمنية وسرعة تنفيذ محددة، وتنقسم هذه الأنظمة إلى 3 أنواع:

Soft Real Time Systems وهي الأنظمة التي تسمح ببعض التأخير المحدود في التنفيذ أو الإستجابة للمهام المطلوب تنفيذها. أمثلة على ذلك، الهواتف النقالة، مشغلات الألعاب، أجهزة عرض الفيديو. كل ما سبق أنظمة يمكن التغاضي عن حدوث تأخير زمني بسيط أثناء تشغيلها. فمثلاً لن تحدث مشكلة إذا تأخرت شاشة الهاتف النقال بضعة مللي ثانية في الاستجابة لأوامر المستخدم.

Hard Real Time Systems وهي الأنظمة التي دائماً ما يجب أن تنجح في تنفيذ جميع المهام الخاصة بها في الزمن المطلوب وبدون أي تأخير في الإستجابة مهما كان. وحدوث أي تأخير يعني فشل النظام ككل. وقد يؤدي إلى كارثة. من الأمثلة على هذه الأنظمة وحدات التحكم في المفاعلات النووية، الصواريخ، الطائرات الحربية، بعض الأجهزة الطبية مثل أجهزة تنظيم ضربات القلب.

Firm Real Time Systems تعتبر مماثلة لـ Hard Real Time systems من ناحية سرعة الإستجابة والتنفيذ لكن الاختلاف هو أنه في حالة الفشل يُضطر مرات للاستجابة المطلوبة فلن يؤدي ذلك إلى فشل النظام ككل. ومثال على ذلك جهاز الراوتر اللاسلكي **Wireless Router** (الذي يبث الإنترن特 بصورة لاسلكية للهواتف أو الأجهزة المحمولة) يحتوي على شريحة تعمل على إستقبال وإرسال حزم البيانات **Data packets** باستجابة عالية ومماثلة لكن إذا حدث أن قدرت بعض الحزم أو لم يستطع الراوتر التعامل معها فإن ذلك لا يعني فشل للنظام ككل فهذا يعتبر خطأ قبول طالما أنه ليس متكرر بكثرة.



11.2 طرق تصميم الـ Real Time Embedded systems

هناك العديد من الطرق لتصميم نظم مدمجة عالية الإستجابة بعضها يتسم بالبساطة والبعض الآخر مصمم ليعالج الوظائف المعقّدة والمتدخلة. من هذه الطرق هناك نوعين أساسين:

الطريقة الأولى: كتابة الوظائف التي لا تحتاج استجابة سريعة داخل الدالة الرئيسية Main function بينما المهام التي تتطلب استجابة لحدث خارجي معين أو تحدث بصورة دورية يتم وضعها داخل دوال مقاطعات .interrupts.

```
#include "library1"
#include "library2"
int main()
{
    some_data

    while(1)
    {
        هنا تكتب المهام ذات الأهمية التقليدية // 
    }
    return 0;
}
```

```
ISR(type_of_ISR)
{
    كود معالجة سريع لأحد المهام عند حدوث مقاطعة //
}
```

```
ISR(type_of_ISR)
{
    كود معالجة سريع لأحد المهام عند حدوث مقاطعة //
}
```

يعيب هذه الطريقة أنها لا تصلح إلا لعدد محدود جداً من المهام كما أن كتابة الكود البرمجي لأكثر من مهمة داخل الدالة الرئيسية يجعل الكود معقد.

الطريقة الثانية: استخدام أنظمة تشغيل الوقت الحقيقي RTOS والتي تختصر بكلمة هذه الأنظمة مشابه في نظرية عملها بنظام التشغيل التقليدي الذي تستخدمه الأن على الحاسوب الآلي مثل Windows أو Linux.

تهدف أنظمة التشغيل إلى توفير مجموعة من الخدمات أهمها "تعدد المهام Multi-tasking" بفضل هذه الأنظمة تستطيع أن تتصفح الويب وفي نفس الوقت سماع ملف صوتي وقد تستخدم برنامج معالجة النصوص بجانب كل ما سبق.

بالنسبة لك أنت تستطيع أن تقوم بكل هذه الأشياء في نفس الوقت لكن الحقيقة أن هذه المهام تتم بالتتابع، أما سرعة تنفيذها فيرجع الفضل في ذلك إلى مهارة نظام التشغيل في معالجة كل هذه المهام بسرعة واستجابة عالية.



كيف تعمل أنظمة التشغيل بشكل عام؟

يعمل نظام التشغيل بصورة مشابهة للطباخ المحترف، عادة يطلب من الطباخ أن يعد أكثر من وجبة في نفس الوقت وقد تختلف هذه الوجبات أو تتشابه لكن في النهاية يتوجب عليه إعدادها جميعاً في وقت محدد.



عادة ما يلجن الطباخ لحيلة ذكية لتنظيم الوقت فهو لا يقوم بإنها كل جزء من الوجبة على حدة ثم البدأ في جزء آخر ولكنه بدلاً من ذلك يقوم بعملية التبديل بين المهام Task switching. فمثلاً إذا كان المطلوب هو إعداد طبق المعكرونة مع اللحم المشوي فإن الطباخ سيدأ أولأ بغلة الماء ويتركه على النار ثم قد يضع اللحم على الشواية (المهمة الثانية) ثم يبدأ في تجهيز الطماطم من أجل الصلصة (المهمة الثالثة)، عندما يستشعر أن الماء بدأ يغلي بدرجة حرارة مناسبة سيترك الصلصة ثم يعود إلى الماء ليضيف إليه المعكرونة ويتركها حتى تتصبح، ثم يعود مرة أخرى للطماطم و في ذلك الوقت فإنه سيحاول كل فترة زمنية أن يطمئن على درجة نضوج اللحم وتستمر هذه العملية حتى تنتهي كل مكونات الوجبة ...

نظام التشغيل يعمل بصورة مشابهة، في البداية يقوم بتجهيز قائمة بالمهمات التي يجب عليه أن ينجذبها ثم يبدأ بإدارة المهام مثل الطباخ المحترف، ويقوم بذلك بعده طرق منها مثلاً أن يعطي كل مهمة فترة معالجة زمنية قصيرة تسمى time slice (شريحة زمنية) وعند انتهاء الشريحة الزمنية يقوم بالتبديل بين المهام. ويعرف هذا الأسلوب لجدولة المهام بالـ Round Robin scheduling

أيضاً قد يقوم نظام التشغيل بمعالجة المهام على حسب أولوية إنهائها فمثلاً قد يبدأ في معالجة أهم مهمة متوفرة لفترة طويلة حتى تصله رسالة أو طلب مقاطعة بوجود مهمة أخرى أعلى أهمية وتحتاج لمعالجة فورية فيترك المهمة الأولى ويدهش للثانية مؤقتاً حتى ينهيها ثم يعود للأولى. وعندما ينتهي من المهمة الأولى بالكامل يتركها ويدهش إلى المهمة الثالثة وهكذا ... ويعرف هذا الأسلوب في التبديل بين المهام بالـ .Prioritiry based scheduling

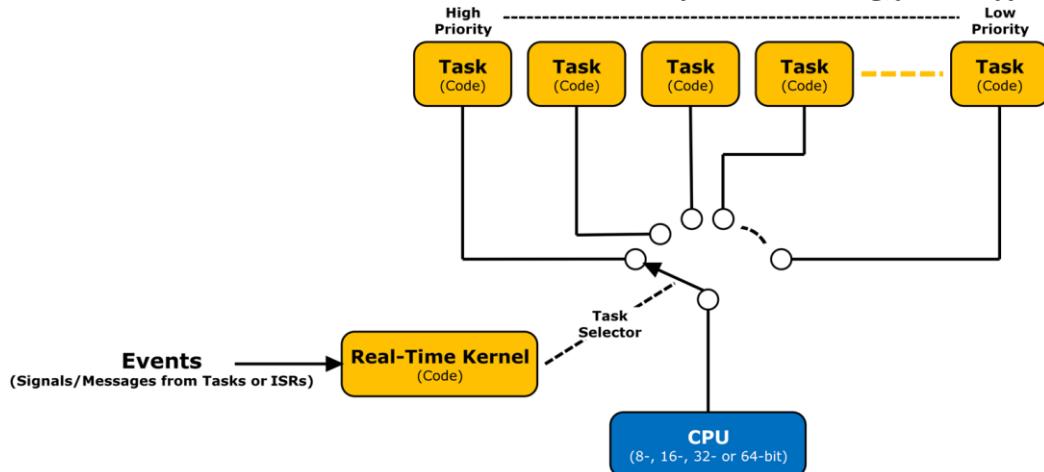
و قبل أن نكمل هناك بعض المسميات الإنجليزية الواجب معرفتها:

يسمى الجزء الذي يدير عملية التبديل بين المهام بالـ RTOS scheduler (مجدول نظام التشغيل) وهو جزء من نواة



النظام والتي تسمى RTOS kernel كما تسمى عملية التبديل بين المهام (بعض النظر عن أسلوب جدولة التبديل) باسم Context switching

وتحتاج المهمة التي يتم معالجتها الآن بالـ **Running task** بينما المهمة التي تنتظر دورها في المعالجة **Waiting task**. الصورة التالية توضح طريقة عملRTOS بأسلوب مبسط



11.3 كيف تعمل النواة RTOS Kernel

المعالجات الموجودة داخل المتحكمات الدقيقة تستطيع أن تنفذ مجموعة من الأوامر المتتالية ولا تعرف تحديداً إذا كانت هذه الأوامر جزء من نظام التشغيل أو أنها مهمة معينة وهذا يؤدي إلى سؤال هام، كيف يفهم المعالج أنه يجب أن يترك المهمة الحالية ويدرك لتنفيذ مهمة أخرى؟

هنا تأتي وظيفة نواة النظام Kernel. هذه النواة تعمل داخل دالة مقاطعة زمنية Interrupt service routine (ISR) ويتم تكرار تشغيلها من مئات إلى آلاف المرات في الثانية الواحدة (يسمى عدد مرات تشغيل النواة بالـ System tick). في كل مرة يتم تشغيل النواة يبدأ جزء من هذه النواة يسمى "برنامجه الجدوله Scheduler" بمراجعة قائمة المهام الموجودة ويحدد ما الذي سيتم معالجته الآن.

قد يختار برنامج الجدوله أن يكمل المهمة الحالية أو ينتقل إلى مهمة أخرى على حسب خوارزمية التحويل (ـ Round Robin أو Priority-based) وبعد أن يقرر المهمة يقوم بنقل البيانات الخاصة بمعالجة هذه المهمة إلى المعالج وتتضمن البيانات محتوى المسجلات المختلفة (كل من المسجلات العامة والخاصة).



FreeRTOS 11.4 مقدمة عن نظام

تتوفر العديد من أنظمة تشغيل الوقت الفعلي المجانية والتي تختلف فيما بينها بما تقدمه من إمكانيات ووظائف إضافية، من هذه الأنظمة نجد FreeRTOS والذي يعد أشهر أنظمة الوقت الحقيقي مفتوحة المصدر المتوفرة للإستخدام المجاني سواء بصورة تعليمية أو تجارية.

يدعم هذا النظام مختلف المُتحكمات الدقيقة فيمكنك تشغيله على PIC – ARM – AVR – 8051 – PowerPC – x86



كما يوجد منه نسخة مخصصة للأنظمة ذات الموثوقية العالية (مثل الأجهزة الطبية أو التطبيقات العسكرية) ويسمى SAFE-RTOS مع العلم أن هذه النسخة مدفوعة وليس مجانية.



يمتلك هذا النظام العديد من المميزات الرائعة منها:

- حجم صغير يستهلك أقل من 4 إلى 9 كيلو بايت من ذاكرة الـ ROM مما يجعله مناسب لمعظم المُتحكمات الدقيقة (تذكر أن ATmega16 يمتلك 16 كيلوبايت من الذاكرة بينما يمتلك ATmega32 نحو 32 كيلوبايت).
- توفير مجموعة من المكتبات البرمجية الجاهزة للتعامل مع أنظمة الملفات FAT والـ storage media مثل بطاقات الذاكرة.
- دمج معه مكتبات خاصة لتسهيل معالجة البيانات القادمة من وإلى شبكات الحاسب الآلي عبر بروتوكول TCP/IP أو UDP مما يجعله نظام مناسب لتطبيقات إنترنت الأشياء IoT والتحكم عن بعد.



11.5 الهيكل البرمجي لـ RTOS

بصورة افتراضية تكتب جميع أوامر برامج النظم المدمجة في الدالة الرئيسية على عكس أنظمة RTOS حيث نجد أن كل مهمة task تماثل دالة رئيسية مستقلة وتكون مهمة Main Function هيتعريف عدد وخصائص المهام فقط.

الهيكل التالي شكل التقليدي للبرامج المكتوبة باستخدام RTOS حيث تم إنشاء 3 مهام مختلفة وتحتكر كل مهمة مجموعة الأوامر الخاصة بها.

```
#include "RTOS.h"
```

```
int main()
{
    createTask( task1, task1_parameters);           // إنشاء المهمة الأولى
    createTask( task2, task2_parameters);           // إنشاء المهمة الثانية
    createTask( task3, task3_parameters);           // إنشاء المهمة الثالثة

    startRTOS_scheduler();                         // إبدء إدارة المهام باستخدام RTOS

    while(1);                                    // لا تفعل أي شيء آخر داخل الدالة الرئيسية إلى ما لا نهاية
    return 0;
}
```

```
void task1() {

    while(1) {
        هنا تكتب مجموعة الأوامر الخاصة بالمهمة الأولى
    }
}
```

```
void task2() {

    while(1) {
        هنا تكتب مجموعة الأوامر الخاصة بالمهمة الثانية
    }
}
```

```
void task3() {

    while(1) {
        هنا تكتب مجموعة الأوامر الخاصة بالمهمة الثالثة
    }
}
```



11.6 تشغيل FreeRTOS على جميع مُتحكمات AVR

عند تحميل نظام FreeRTOS من الموقع الرسمي سنجد أنه مهيأ للعمل على المُتحكم الدقيق **atmega323** فقط. إذا سنقوم بعمل ما يسمى "تصدير النظام" **system porting** (البعض يسميه تهيئة النظام)، والتي تعني ضبط النظام ليعمل مع مختلف المُتحكمات الدقيق الأخرى من عائلة AVR. وسنأخذ المُتحكمات **16/32/328/644/1284** مثلاً على ذلك.

في البداية قم بتحميل أحدث إصدار متوفّر من نظام FreeRTOS (وقت كتابة هذا الفصل كان الإصدار رقم 8.2.1) من الموقع الرسمي <http://www.freertos.org>

The screenshot shows the official FreeRTOS website. At the top, there's a navigation bar with links for Quality RTOS & Embedded Software, About, Contact, Support, and FAQ. Below the navigation is a main content area. On the left, there's a sidebar with links for Quick Start Guide + Videos, Front Page (homepage), Download (with sub-links for FreeRTOS Books and Manuals, FreeRTOS Interactive!, and Contact, Support, Advertising), and FreeRTOS+ Ecosystem (with sub-links for Internet of Things, InterNiche TCP/IP, and FreeRTOS BSPs). The main content area features a section titled "FreeRTOS™" with a sub-section "The Market Leading, De-facto Standard and Cross Platform Real Time Operating System (RTOS). Don't Let Your RTOS Lock You In." It also mentions that FreeRTOS is developed in partnership with leading chip companies over a 12-year period. To the right, there's a "Latest News" section with a link to "FreeRTOS V8.2.1 Released" and a "New: FreeRTOS Labs" section with a link to "New in the FreeRTOS Lab".

اضغط على **Download** من الجانب الأيسر للموقع للانتقال لصفحة التحميل ثم اختر "تحميل الإصدار الأخير من موقع sourceforge.com" كما في الصورة التالية

The screenshot shows the "RTOS Source Code Download Instructions" page on SourceForge. It starts with a heading "RTOS Source Code Download Instructions" and a sub-section "To download, and get the most out of FreeRTOS, follow the following steps:". Below this, there are three numbered steps: 1. Please keep up to date by joining the [announcements mailing list](#) for infrequent comprehensive notifications. 2. The download is available as a standard zip file (.zip), and as a self extracting zip file (.exe). 3. Unzip the source code into a suitable directory - taking care to ensure the directory structure within the zip file is maintained. At the bottom of the page, there's a link "Click to download the latest official release from SourceForge".

عند الانتهاء من التحميل قم بفك ضغط الملف لنجد المجلد الرئيسي باسم FreeRTOS وبه جميع المجلد والملفات الخاصة بنظام التشغيل ل مختلف المُتحكمات الدقيقة. تنقسم الملفات إلى مجلدين أساسين وهما **source** و **Demo**. الأول يحتوي على جميع الملفات المصدرية لنظام التشغيل التشغيل نفسه بينما الآخر يحتوي على أمثلة لاختبار نظام التشغيل.

ملاحظة: الملفات تشمل نسخة من النظام + أمثلة لجميع المُتحكمات من جميع العائلات التي يدعمها نظام FreeRTOS بما في ذلك مُتحكمات PIC, AVR, ARM cortex ما يهمنا من هذه الملفات ما هو متعلق بالـ AVR فقط باعتباره موضوع الكتاب.

أيضاً يدعم نظام FreeRTOS العديد من المترجمات المختلفة مثل Keil - IAR - GCC - و لكننا سنستخدم GCC فقط باعتباره المترجم المرفق مع الـ **toolchain** المجانية.



تعتمد عملية تصدر النظام على التلاعب بإعدادات FreeRTOS من خلال 3 ملفات رئيسية:

port.c
FreeRTOSConfig.h
makefile
main.c

تتوارد هذه الملفات في المسارات التالية:

FreeRTOS/Source/portable/GCC/ATMega323/port.c
 FreeRTOS/Demo/AVR_ATMega323_WinAVR/FreeRTOSConfig.h
 FreeRTOS/Demo/AVR_ATMega323_WinAVR/makefile

أولاً: تعديل **port.c**

سمى هذا الملف **port.c** باعتباره الملف الأساسي في عملية الـ porting حيث يتحكم بالخصائص الأساسية لعملية الـ (تبديل بين المهام tasks). ويتم ضبط إعدادات المؤقت والمقطاعات الدورية من داخل الملف.

يعتمد نظام FreeRTOS على المؤقت رقم 1 في جميع عائلات AVR وبالتحديد يقوم بالتنقل بين المهامات عندما يحدث مقاطعة نتيجة تطابق عداد المؤقت 1 والمسجل OCRA1x و هو ما يعرف باسم timer 1 COMPARE A match .interrupt

في البداية توجه إلى الملف المتواجد داخل المسار التالي

FreeRTOS/Source/portable/GCC/ATMega323/port.c

افتح الملف بأي محرر نصوص متوفّر لديك وتوجه إلى السطر الذي يحتوي العبارة التالية

#define portCOMPARE_MATCH_A_INTERRUPT_ENABLE ((uint8_t) 0x10)

```

83 #include "task.h"
84 /*
85  * Implementation of functions defined in portable.h for the AVR port.
86  */
87
88 /* Start tasks with interrupts enables. */
89 #define portFLAGS_INT_ENABLED ( ( StackType_t ) 0x80 )
90
91 /* Hardware constants for timer 1. */
92 #define portCLEAR_COUNTER_ON_MATCH ( ( uint8_t ) 0x08 )
93 #define portPRESCALE 64 ( ( uint8_t ) 0x03 )
94 #define portCLOCK_PRESCALER ( ( uint32_t ) 64 )
95 #define portCOMPARE_MATCH_A_INTERRUPT_ENABLE ( ( uint8_t ) 0x10 )
96
97 /*
98 */

```

أول ما يجب ضبطه هو قيمة الـ mask الذي سيفعل خاصية المقاطعة timer 1 compare match A ويتم حساب هذا الـ mask بالنظر إلى المسجل AVR في متحكمات TIMSK0 القديمة نسبياً أو TIMSK0 في المتحكمات الأحدث، يحتوي هذا المسجل على البت المطلوب تفعيلها **OCIE1A** والمسؤولة عن تفعيل المقاطعة Timer1 Compare A Match interrupt

القيمة 100x تمثل قيمة الـ mask المتحكم في تفعيل المقاطعة ويتم حساب هذه القيمة كالتالي:



- توجه إلى ملف البيانات الخاصة بالمحكم الذي يتضمن عليه نظام التشغيل (على سبيل المثال المتحكم 4 – صفحة دليل البيانات رقم 82). سنجد أن البت رقم 4 OCIE1A هي البت رقم ATmega32

Timer/Counter Interrupt Mask Register – TIMSK	Bit	7	6	5	4	3	2	1	0	TIMSK
	Read/Write	R/W								
	Initial Value	0	0	0	0	0	0	0	0	

• Bit 1 – OCIE0: Timer/Counter0 Output Compare Match Interrupt Enable

- الآن سنقوم بحساب الـ mask وذلك عن طريق وضع الرقم 1 مكان هذه البت وبباقي البتات = 0 وهو ما يعني الرقم 00010000 بالصيغة الثنائية وهو ما يساوي $100x$ بصيغة الهايكس. (هذه القيمة هي الافتراضية بالفعل)

في حالة المتحكم OCIE1A أو atmega328 atmega168 (ومعظم المتحكمات الحديثة) سنجد أن مكان البت مختلف. فمثلاً في حالة المتحكم atmega328 سنجد الوضع التالي (صفحة 135 من دليل البيانات للمتحكم atmega328).

TIMSK1 – Timer/Counter1 Interrupt Mask Register									
Bit	7	6	5	4	3	2	1	0	TIMSK1
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

في حالة المتحكم atmega1284 أو المتحكم atmega644 صفحة 134 من دليل البيانات

TIMSK1 – Timer/Counter1 Interrupt Mask Register									
Bit	7	6	5	4	3	2	1	0	TIMSK1
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

إذا قمنا بوضع الرقم 1 مكان هذه البت وبباقي البتات = صفر سنجد أن قيمة الـ mask تساوي 00000010 وهو ما يساوي $100x$ بصيغة الهايكس. إذا يجب استبدال القيمة 0020x بالقيمة 020x كالتالي:

```
#define portPRESCALE_64          ( ( uint8_t ) 0x03 )
#define portCLOCK_PRESCALER       ( ( uint32_t ) 64 )
#define portCOMPARE_MATCH_A_INTERRUPT_ENABLE ( ( uint8_t ) 0x02 )
```

لاحظ أن المتحكمات الحديثة نسبياً من عائلة AVR – 8bit لا تمتلك المسجل TIMSK0 وإنما يكون باسم TIMSK1 كما في حالة atmega328 و atmega1284

بعد تعديل قيمة الـ mask سنقوم بتعديل بعض أسماء المسجلات في نفس الملف وبالتحديد في الدالة **prvSetupTimerInterrupt(void)** والتي ستجدها في آخر الملف port.c كما في الصورة التالية:



```

404 static void prvSetupTimerInterrupt( void )
405 {
406     uint32_t ulCompareMatch;
407     uint8_t ucHighByte, ucLowByte;
408
409     /* Using 16bit timer 1 to generate the tick.  Correct fuses must be
410      selected for the configCPU_CLOCK_HZ clock. */
411
412     ulCompareMatch = configCPU_CLOCK_HZ / configTICK_RATE_HZ;
413
414     /* We only have 16 bits so have to scale to get our required tick rate. */
415     ulCompareMatch /= portCLOCK_PRESCALER;
416
417     /* Adjust for correct value. */
418     ulCompareMatch -= ( uint32_t ) 1;
419
420     /* Setup compare match value for compare match A.  Interrupts are disabled
421      before this is called so we need not worry here. */
422     ucLowByte = ( uint8_t ) ( ulCompareMatch & ( uint32_t ) 0xff );
423     ulCompareMatch >>= 8;
424     ucHighByte = ( uint8_t ) ( ulCompareMatch & ( uint32_t ) 0xff );
425     OCR1AH = ucHighByte;
426     OCR1AL = ucLowByte;
427
428     /* Setup clock source and compare match behaviour. */
429     ucLowByte = portCLEAR_COUNTER_ON_MATCH | portPRESCALE_64;
430     TCCR1B = ucLowByte;
431
432     /* Enable the interrupt - this is okay as interrupt are currently globally
433      disabled. */
434     ucLowByte = TIMSK;
435     ucLowByte |= portCOMPARE_MATCH_A_INTERRUPT_ENABLE;
436     TIMSK = ucLowByte;
437 }

```

إذا كان المسجل المسؤول عن المقاطعة يسمى **TIMSK** فلن تغير أي شيء بها (مثل المُتحكم 16/32 ATmega32/16) أما إذا كنت تستخدم أي مُتحكم حديث مثل atmega1284 أو atmega328 يجب أن تستبدل TIMSK بالاسم المناسب مثل TIMSK0 أو TIMSK1 (على حسب ما هو مذكور في دليل البيانات كما شاهدنا في الخطوة السابقة).

مثال: تعديل الدالة لتناسب مع المُتحكم atmega1284/644/328/168/....etc

```

432     /* Enable the interrupt - this is okay as interrupt are currently globally
433      disabled. */
434     ucLowByte = TIMSK1;
435     ucLowByte |= portCOMPARE_MATCH_A_INTERRUPT_ENABLE;
436     TIMSK1 = ucLowByte;
437 }
438 /* */

```

الجزء الأخير من تعديل ملف port.c هو إعادة تسمية الـ ISR الخاصة بمقاطعة Timer1 Compare A Match وهذا الجزء يجب تغييره لمعظم مُتحكمات AVR (سواء القديمة أو الحديثة). هذا الجزء ستجده في نهاية ملف port.c كما في الصورة التالية.



```

439
440 #if configUSE_PREEMPTION == 1
441
442 /*
443  * Tick ISR for preemptive scheduler. We can use a naked attribute as
444  * the context is saved at the start of vPortYieldFromTick(). The tick
445  * count is incremented after the context is saved.
446  */
447 void SIG_OUTPUT_COMPARE1A( void ) __attribute__ ( ( signal, naked ) );
448 void SIG_OUTPUT_COMPARE1A( void )
449 {
450     vPortYieldFromTick();
451     asm volatile ( "reti" );
452 }
453 #else
454
455 /*
456  * Tick ISR for the cooperative scheduler. All this does is increment the
457  * tick count. We don't need to switch context, this can only be done by
458  * manual calls to taskYIELD();
459  */
460 void SIG_OUTPUT_COMPARE1A( void ) __attribute__ ( ( signal ) );
461 void SIG_OUTPUT_COMPARE1A( void )
462 {
463     xTaskIncrementTick();
464 }
465 #endif

```

قم بتغيير كل الدوال باسم **SIG_OUTPUT_COMPARE1A** إلى الاسم **TIMER1_COMPA_vect**

ليصبح الملف كالتالي:

```

447 void TIMER1_COMPA_vect( void ) __attribute__ ( ( signal, naked ) );
448 void TIMER1_COMPA_vect( void ) __attribute__ ( ( signal ) );
449 {
450     vPortYieldFromTick();
451     asm volatile ( "reti" );
452 }
453 #else
454
455 /*
456  * Tick ISR for the cooperative scheduler. All this does is increment the
457  * tick count. We don't need to switch context, this can only be done by
458  * manual calls to taskYIELD();
459  */
460 void TIMER1_COMPA_vect( void ) __attribute__ ( ( signal ) );
461 void TIMER1_COMPA_vect( void )
462 {
463     xTaskIncrementTick();
464 }
465 #endif

```

بذلك نكون قد انتهينا من تعديل الملف **port.c**



ثانياً: تعديل FreeRTOSConfig.h

يتتحكم هذا الملف في إعدادات نظام التشغيل عندما يبدأ العمل على المتحكم الدقيق، وبالتحديد هو المسؤول عن التلاعب بخصائص الـ kernel وبعض خصائص تنظيم المهام tasks. يمكنك استخدام الملف كما هو بالوضع الافتراضي لكن ستحتاج أن تغير بعض الإعدادات الهامة لتتناسب مع المتحكم الدقيق المستخدم. وبعض الإعدادات الأخرى والتي ستجعل نظام التشغيل أكثر كفاءة في إدارة المهام tasks خاصة مع المتحكمات ذات الذاكرة الصغيرة.

يتواجد الملف في نفس مجلد main.c بجانب الملف main.c وبالتحديد في المسار التالي
FreeRTOS/Demo/AVR_ATMega323_WinAVR/FreeRTOSConfig.h

الإعدادات الواجب ضبطها

#define configCPU_CLOCK_HZ ((unsigned long) 8000000)

يتتحكم هذا الخيار في ضبط التوقيت داخل نظام التشغيل، يجب أن تغير قيمته إلى نفس التردد الذي يعمل به المعالج (القيمة الافتراضية هي 8 ميجاهرتز).

#define configTICK_RATE_HZ ((TickType_t) 1000)

عدد المرات التي تعمل بها الـ kernel وذلك التبديل بين المهامات المختلفة. فمثلاً بافتراض أن لديك 10 مهامات فهذا يعني أن الـ Kernel يقوم بالتبديل بينهم 1000 مرة وهو ما يعني أن كل مهمة سيتم زيارتها 100 مرة في الثانية في الواحد.

زيادة هذا الرقم ستؤدي إلى نقص الوقت المخصص للمعالجة لكل مهمة في مقابل تحسين الاستجابة للمهامات كل، ولكن هناك عيب واحد وهو أن الـ kernel سيزداد معدل استهلاكها للوقت والموارد بزيادة هذا الرقم. فمثلاً إذا كان معدل الـ tick = 3000 Hz ستعمل 3000 مرة في الثانية وفي كل مرة قد تختار أحد المهامات لتشغيلها أو context switch مع مهمة أخرى.

من المفید زيادة هذا الرقم في حالة أن المهامات تحتاج لاستجابة سريعة لكن يستحسن أن يكون المتحكم يعمل بسرعة 16 أو 20 ميجاهرتز على الأقل. مع العلم أنه بزيادة هذا الرقم سيتم استهلاك المزيد من الطاقة. لذا إن كان مشروعك يجب تصميمه باستهلاك طاقة منخفض فيستحسن تقليل الرقم إلى 500 (سيخفض سرعة الاستجابة للمهامات) وإذا كنت لا تهتم بزيادة استهلاك الطاقة فيمكنك أن تضع القيمة بين 1000 و 3000 وفي حالة المتحكمات من عائلة AVR أو AVR Xmega أو 32 bit يمكن زيادة الرقم إلى 5000.

#define configMAX_PRIORITIES (4)

هذا الخيار يتحكم بأقصى عدد من الأولويات المسموح بها في نظام التشغيل. يتم تحديده بناء على عدد المهامات المتوقع تشغيلها على النظام ومدى أهمية كل منها. إذا كانت معظم المهامات لها نفس الأولوية فيمكنك أن تضع الرقم ب 2 فقط.

#define configMINIMAL_STACK_SIZE ((unsigned short) 85)

الرقم المسؤول عن تحديد أقل حجم للـ stack لكل مهمة تعمل، يجب أن يتناسب الرقم مع حجم الذاكرة العشوائية SRAM للمتحكم الدقيق. فمثلاً إذا كانت الـ SRAM تساوي 1 كيلو فمن الأفضل أن يصبح الرقم 50 فقط أما إذا كانت الذاكرة 2 كيلو فيمكنك أن تتركه كما هو أما إذا كانت الذاكرة 4 كيلو أو أكثر فمن الممكن زيتها إلى 150.

أيضاً يجب زيادة هذا الرقم إذا كانت المهامات التي سيتم تشغيلها ستعمل مع دوال أو عمليات حسابية كبيرة (أو أي عملية تحتاج أن تستخدم الـ Stack لإجرائها). ولاحظ أن معظم العمليات الحسابية المركبة مثل $y = 5 + z * (x + 1)$ تتطلب stack لذا يجب أن تضع هذه العوامل بالحساب عند اختيار هذا الرقم. أيضاً يتم استخدام الـ stack في حفظ قيم المسجلات التي تتعامل معها المهمة.



يجب الانتباه إلى أمر `HAM` وهو أن حجم الـ `stack` لكل مهمة يقاس بالـ `word` وليس بالـ `byte` لذا عندما نقول أن حجم الـ `stack` = 150 فهذا يعني أن الحجم الحقيقي في الذاكرة $stack\ size = 150\ word = 150 * (4\ byte\ "1\ Word") = 600\ byte$

```
#define configTOTAL_HEAP_SIZE ( (size_t) ( 1500 ) )
```

الـ `Heap` هي تقنية `data-type` تستخدم أنظمة RTOS لتخفيص ذاكرة مرننة (قابلة للزيادة) لكل مهمة تعمل في نظام التشغيل. يتم استخدام هذه التقنية في حفظ جميع المتغيرات أو الثوابت المستخدمة في كل مهمة.

يتم تحديد هذا الرقم بصورة أساسية على حسب حجم الذاكرة العشوائية الكلية للمتحكم الدقيق ويجب الانتباه أن الرقم 1500 يساوي قيمة الذاكرة بالبايت `byte` وهذا يعني أن ذاكرة الـ `heap` هنا تساوى الخامس كيلو بايت تقريباً.

يمكن تحديد هذا الرقم بما هو إجمالي الذاكرة العشوائية للمتحكم الدقيق مع مراعاة أن نظام freeRTOS يحتاج نحو 300 إلى 400 بايت لتشغيل الـ `kernel` بينما يمكن تخصيص باقي الذاكرة للمهامات المختلفة.

الأمثلة التالية هي لقيم الـ `heap` للمتحكمات المختلفة

- المتحكم **ATmega16** يمتلك SRAM = 1 Kbyte
- مساحة الـ `heap` يفضل أن تكون **0.8 كيلوبايت (800)**

- المتحكمات **ATmega32/328** يمتلك SRAM = 2 Kbyte
- مساحة الـ `heap` يفضل أن تكون **واحد ونصف كيلوبايت (1500)**

- المتحكم **atmega644** يمتلك SRAM = 4 Kbyte
- مساحة الـ `heap` يفضل أن تكون **3.5 كيلوبايت (3584)**

- المتحكم **atmega1284** يمتلك SRAM = 16 Kbyte
- مساحة الـ `heap` يفضل أن تكون **15.5 كيلوبايت (15360)**

يمكنك أن تتعلم حول باقي الإعدادات من المرجع المفصل لملف `FreeRTOSConfig.h` على الرابط التالي <http://www.freertos.org/a0019.html>

ثالثاً: تعديل `serial.c` (اختياري - غير مطلوب)

يوفر نظام FreeRTOS ملف `serial.c` وهو `driver` لتشغيل الـ `UART` داخل نظام التشغيل الـ `AVR 8 bit` لكن يحتاج بعض التعديلات البسيطة وهي تغيير أسماء المسجلات الداخلية المستخدمة في الملف (أيضاً بسبب اختلاف المتحكمات الحديثة عن القديمة في تسمية المسجلات مثل `UCSRB` `UCSROB` `UCSR1B` أو `UCSR1B`). التعديلات المقترنة (على حسب المتحكم المستخدم) - يجب مراجعة دليل البيانات الخاص بالـ `USART` لمعرفة أسماء المسجلات



مثال: تعديل الـ **driver** ليتوافق مع المُتحكم **atmega328** (يجب استبدال جميع المُسجّلات بالاسماء الصحيحة لها).

UCSRB -> **UCSROB**
 UCSRC -> **UCSROC**
 UBRRL -> **UBRROL**
 UBRRH -> **UBRROH**
 UDR -> **UDR0**

ملاحظة: لن يتم استخدام هذا الملف في جميع الأمثلة الموجودة في الكتاب والتعديلات المذكورة بالأعلى اختيارية في حالة أنك تريد استخدام الـ **UART** في مشاريعك الخاصة

رابعاً: تعديل **makefile**

التعديلات في هذا الملف مرتبطة بمستخدمي نظام **Linux** أو أي شخص يستخدم الـ **toolchain** **Linux** مباشرة دون استخدام بيئة برمجة متكاملة مثل (IDE) **ATmel studio**، يمكنك قراءة الملحق الخاص بعملية **compiling using makefile** لتتعرف أكثر على هذه التقنية.

يوجد الملف **makefile** في نفس مجلد **Demo** وبحتوي على بيانات المُتحكم والمبرمجة التي سيتم استخدامها، حيث يجب ضبط هذه الإعدادات لتوافق مع المُتحكم المستخدم.

تغيير المُتحكم

في بداية الملف ستجد السطر البرمجي **MCU = atmega323** قم بتغييره لاسم المُتحكم المطلوب مثل **ATmega16** أو **ATmega32** أو **ATmega1284** كما في الصورة التالية

```

33 # MCU name
34 MCU = atmega32
35

```

تعديل المكتبات المُضافة

في الجزء الخاصه بالـ **makefile** سنجد أن ملف **makefile** الافتراضي يتضمن العديد من المجلدات التي تحتوي جميع المكتبات المطلوبة لتشغيل نظام **FreeRTOS**، يمكنك إضافة مكتباتك الخاصة أو مسح المكتبات التي لا تحتاجها من هذا الجزء.



```

46
47 # List C source files here. (C dependencies are automatically generated.)
48 DEMO_DIR = ../../Common/Minimal
49 SOURCE_DIR = ../../Source
50 PORT_DIR = ../../Source/portable/GCC/ATMega323
51
52 SRC = \
53 main.c \
54 ParTest/ParTest.c \
55 serial/serial.c \
56 regtest.c \
57 $(SOURCE_DIR)/tasks.c \
58 $(SOURCE_DIR)/queue.c \
59 $(SOURCE_DIR)/list.c \
60 $(SOURCE_DIR)/croutine.c \
61 $(SOURCE_DIR)/portable/MemMang/heap_1.c \
62 $(PORT_DIR)/port.c \
63 $(DEMO_DIR)/crflash.c \
64 $(DEMO_DIR)/integer.c \
65 $(DEMO_DIR)/PollQ.c \
66 $(DEMO_DIR)/comtest.c
67

```

في الأمثلة المذكورة في الكتاب لا داعي لإضافة كل هذه المكتبات لذا سنقوم بمسحها واستبدلها بالقائمة التالية فقط

```

SRC = \
main.c \
$(SOURCE_DIR)/tasks.c \
$(SOURCE_DIR)/queue.c \
$(SOURCE_DIR)/list.c \
$(SOURCE_DIR)/croutine.c \
$(SOURCE_DIR)/timers.c \
$(SOURCE_DIR)/portable/MemMang/heap_1.c \
$(PORT_DIR)/port.c

```

المكتبات المُضافة بعد التعديل

```

47 # List C source files here. (C dependencies are automatically generated.)
48 DEMO_DIR = ../../Common/Minimal
49 SOURCE_DIR = ../../Source
50 PORT_DIR = ../../Source/portable/GCC/ATMega323
51
52 SRC = \
53 main.c \
54 $(SOURCE_DIR)/tasks.c \
55 $(SOURCE_DIR)/queue.c \
56 $(SOURCE_DIR)/list.c \
57 $(SOURCE_DIR)/croutine.c \
58 $(SOURCE_DIR)/timers.c \
59 $(SOURCE_DIR)/portable/MemMang/heap_1.c \
60 $(PORT_DIR)/port.c
61

```

تعديل المُبرمجة

هنا سنقوم باختيار المُبرمجة المستخدمة في حرق ملف الهيكس، ابحث عن العبارة التالية

AVRDUDE_PROGRAMMER = stk500

شخصياً أستخدم usbasp لذا سأقوم باستبدال stk500 بكلمة usbasp (استبدلها باسم المُبرمجة التي لديك إذا كانت مختلفة)



عن usbasp) كما في الصورة التالية

```

165 AVRDUDE_PROGRAMMER = usbasp
166
167
168 AVRDUDE_PORT = com1      # programmer connected to serial device
169 #AVRDUDE_PORT = lpt1      # programmer connected to parallel port
170
171 AVRDUDE_WRITE_FLASH = -U flash:w:$(TARGET).hex
172 #AVRDUDE_WRITE_EEPROM = -U eeprom:w:$(TARGET).eep
173
174 AVRDUDE_FLAGS = -p $(MCU) -P $(AVRDUDE_PORT) -c $(AVRDUDE_PROGRAMMER)
175

```

ملاحظة: في حالة أنك ستستخدم usbasp يجب أن تمسح الخيار

-P \$(AVRDUDE_PORT)

أما في حالة استخدام أي مبرمجة أخرى تمتلك بورت على نظام ويندوز أو لينكس مثل ISP AVRISP mkII usbTiny ISP فيجب أن تحدد رقم البورت في الخيار المسمى AVRDUDE_PORT

```

173
174 AVRDUDE_FLAGS = -p $(MCU) -P $(AVRDUDE_PORT) -c $(AVRDUDE_PROGRAMMER)
175

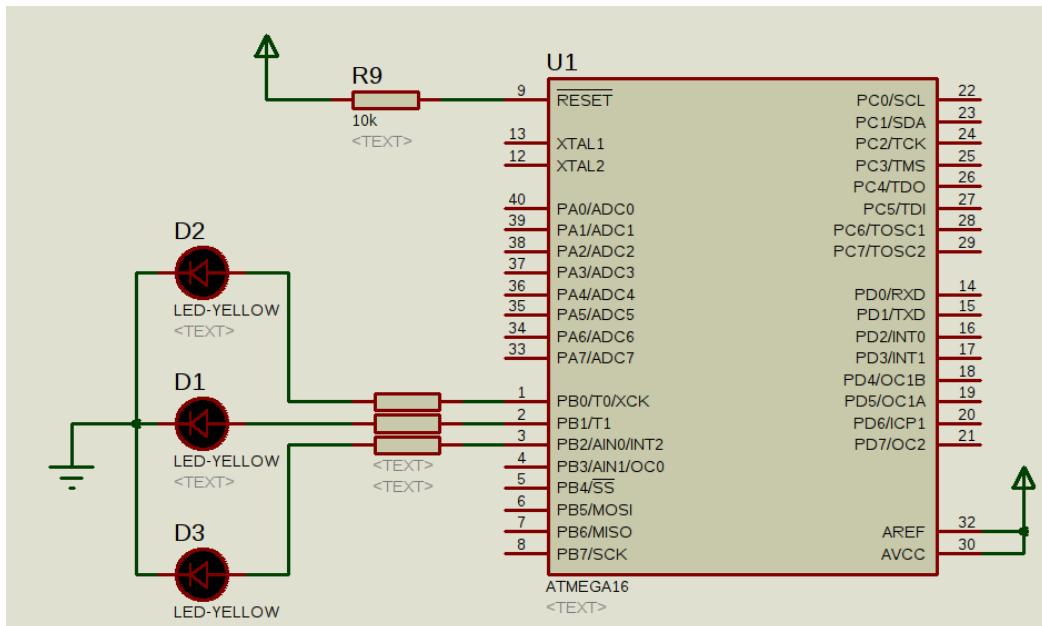
```



11.7 المثال الأول: Blinking 3 leds with 3 tasks

يأتي نظام FreeRTOS بمثال برمجي جاهز لاختبار تعدد المهام لكنه معقد قليلاً لذا أفضل أن نبدأ مع مثال أبسط و أكثر وضوحاً هو عبارة عن تشغيل 3 ليدات مختلفة كل منها تعمل بتقويم مختلف عن الأخرى. الاليدات متصلة بالترتيب على PB0, PB1, PB2 كما هو موضح في الصورة التالية (المثال تم باستخدام ATmega16 و يمكن استخدام ATmega32 بنفس التوصيات أيضاً).

مخطط الدائرة



قم بفتح الملف main.c في مجلد Demo وامسح الأكواد البرمجية الموجودة به واستبدلها بالأكواد التالية

ملاحظة: الأكواد مرفقة في الملف FreeRTOS Examples/blinking_3_leds.c

أيضاً يمكنك استبدال الملف مباشرة بالملف blinking_3_leds.c مع تغيير اسمه إلى

main.c ليصبح هو الملف الرئيسي الذي سيتم ترجمته

الكود

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

/* FreeRTOS files. */
#include "FreeRTOS.h"
#include "task.h"
#include "croutine.h"

/* Define all the tasks */
```



```

static void ledBlinkingtask1(void* pvParameters);
static void ledBlinkingtask2(void* pvParameters);
static void ledBlinkingtask3(void* pvParameters);

int main(void) {

    /* Call FreeRTOS APIs to create tasks, all tasks has the same priority "1" with the same
    stack size */

    xTaskCreate( ledBlinkingtask1, ( signed char * ) "LED1", configMINIMAL_STACK_SIZE,
    NULL, 1, NULL );
    xTaskCreate( ledBlinkingtask2, ( signed char * ) "LED2", configMINIMAL_STACK_SIZE,
    NULL, 1, NULL );
    xTaskCreate( ledBlinkingtask3, ( signed char * ) "LED3", configMINIMAL_STACK_SIZE,
    NULL, 1, NULL );

    // Start the RTOS kernel
    vTaskStartScheduler();

    /* Do nothing here and just run infinite loop */

    while(1){};
    return 0;
}

```

```

static void ledBlinkingtask1(void* pvParameters){

    /* Define all variables related to ledBlinkingtask1*/
    const uint8_t blinkDelay = 50 ;

    /* make PB0 work as output*/
    DDRB |= (1<<PB0);

    /* Start the infinite task 1 loop */
    while (1)
    {
        PORTB ^= (1<<PB0);    //toggle PB0
        vTaskDelay(blinkDelay); //wait some time
    }
}

```



```
static void ledBlinkingtask2(void* pvParameters){

/* Define all variables related to ledBlinkingtask2*/
const uint8_t blinkDelay = 150;

/* make PB1 work as output*/
DDRB |= (1<<PB1);

/* Start the infinte task 2 loop */
while (1)
{
    PORTB ^= (1<<PB1);    //toggle PB0
    vTaskDelay(blinkDelay); //wait some time
}
}
```

```
static void ledBlinkingtask3(void* pvParameters){

/* Define all variables related to ledBlinkingtask3*/
const uint16_t blinkDelay = 600;

/* make PB2 work as output*/
DDRB |= (1<<PB2);

/* Start the infinte task 3 loop */
while (1)
{
    PORTB ^= (1<<PB2);    //toggle PB0
    vTaskDelay(blinkDelay); //wait some time
}
}
```

صورة لملف main.c بعد تعديل الأكواد



```

FreeRTOSConfig.h  x  makefile  main.c  x  serial.c  x  port.c  x
1 #define F_CPU 8000000UL
2 #include <avr/io.h>
3 #include <util/delay.h>
4
5 /* FreeRTOS files. */
6 #include "FreeRTOS.h"
7 #include "task.h"
8 #include "croutine.h"
9
10 /* Define all the tasks */
11 static void ledBlinkingtask1(void* pvParameters);
12 static void ledBlinkingtask2(void* pvParameters);
13 static void ledBlinkingtask3(void* pvParameters);
14
15 int main(void){
16
17     /* Call FreeRTOS APIs to create tasks, all tasks have the same priority "1" with the same stack size*/
18     xTaskCreate( ledBlinkingtask1, ( signed char * ) "LED1", configMINIMAL_STACK_SIZE, NULL, 1, NULL );
19     xTaskCreate( ledBlinkingtask2, ( signed char * ) "LED2", configMINIMAL_STACK_SIZE, NULL, 1, NULL );
20     xTaskCreate( ledBlinkingtask3, ( signed char * ) "LED3", configMINIMAL_STACK_SIZE, NULL, 1, NULL );
21
22     // Start the RTOS kernel
23     vTaskStartScheduler();
24     /* Do nothing here and just run infinite loop */
25
26     while(1){}
27     return 0;
28 }
29

```

بعد الانتهاء من جميع التعديلات احفظ الملف ثم قم بفتح سطر الأوامر في نفس المجلد واتكتب الأمر make لتبدا عملية الترجمة، إذا تمت بنجاح ستظهر نتائج مشابهة للصورة التالية:

```

avr-objcopy: --change-section-lma .eeprom=0x0000000000000000 never used

Creating Extended Listing: rtosdemo.lss
avr-objdump -h -S rtosdemo.elf > rtosdemo.lss

Creating Symbol Table: rtosdemo.sym
avr-nm -n rtosdemo.elf > rtosdemo.sym

Size after:
rtosdemo.elf :
section    size      addr
.data        22    8388864
.text       8000        0
.bss        1651    8388886
.stab      27744        0
.stabstr   13082        0
.comment      17        0
Total       50516

Errors: none
----- end -----

```

الآن يمكنك استخدام ملف الهيكس الناتج من عملية الترجمة سواء في تجربة الـ simulation على برنامج بروتس أو تجربة الملف مباشرة على المتحكم الدقيق ATmega16.

في حالة استخدام متحكم آخر مثل ATmega32 أو ATmega328 لا تنسى أن تغير الإعدادات الخاصة بالملف



– كما هو مذكور في الخطوات في بداية الفصل makefile

ملاحظة: بالرغم من إمكانية تصميم الأنظمة الـ Firm Real time أو Hard Real time مع RTOS إلا أنه عادة ما يتم استخدام تقنية الـ ASIC أو الـ FPGA لتصميم هذه الأنظمة حيث تتمتع هذه التقنيات بالقدرة على معالجة العديد من المهام المختلفة والاستجابة الفائقة لها جميعاً في نفس الوقت. بينما المُتحكمات الدقيقة لا تستطيع أن تشغّل أكثر من مهمة في نفس اللحظة حتى وإن كان باستطاعتها التبديل بين المهامات باستخدام الـ RTOS وهذا لأن أغلب المُتحكمات الدقيقة تمتلك نواة معالجة واحدة Single CPU core



صفحة جديدة
.....



12. المُلحقات الإضافية



- | | |
|--|---|
| تنصيب واستخدام برنامج CodeBlocks كبديل لـ ATmel Studio | ✓ |
| ترجمة الملفات باستخدام MakeFile | ✓ |
| رفع ملف Hex على المتحكم الدقيق | ✓ |
| كيف تستخدم لوحة آردوينو لتعلم برمجة الـ AVR | ✓ |

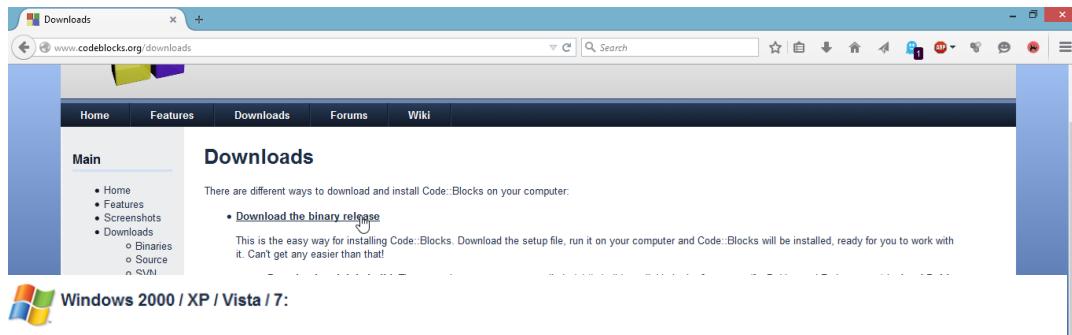


ملحق: تنصيب برنامج CodeBlocks على نظام ويندوز

يعتبر برنامج CodeBlocks من أفضل بيئة البرمجة المخصصة للمشاريع التي تكتب بلغة السي أو السي ++ كما أنه مجاني ومتوفّر لجميع أنظمة التشغيل (ويندوز - لينكس - ماك)، ويعتبر بدلاً أخف وأسرع بكثير من ATmel Studio ومناسب لكل من يريد أن يكتب برمج بلغة السي عامة سواء للمتحكمات الدقيقة أو للحواسيب.

يعتمد البرنامج بصورة أساسية على المترجم الشهير GCC والذي يستخدم الإصدارة المنشورة منه وهي avr-gcc سري في الخطوات القادمة. في البداية قم بتحميل ملفات التنصيب الخاصة بالبرنامج عبر التوجّه إلى موقع CodeBlocks الرسمي من خلال الرابط التالي:

<http://www.codeblocks.org/downloads>



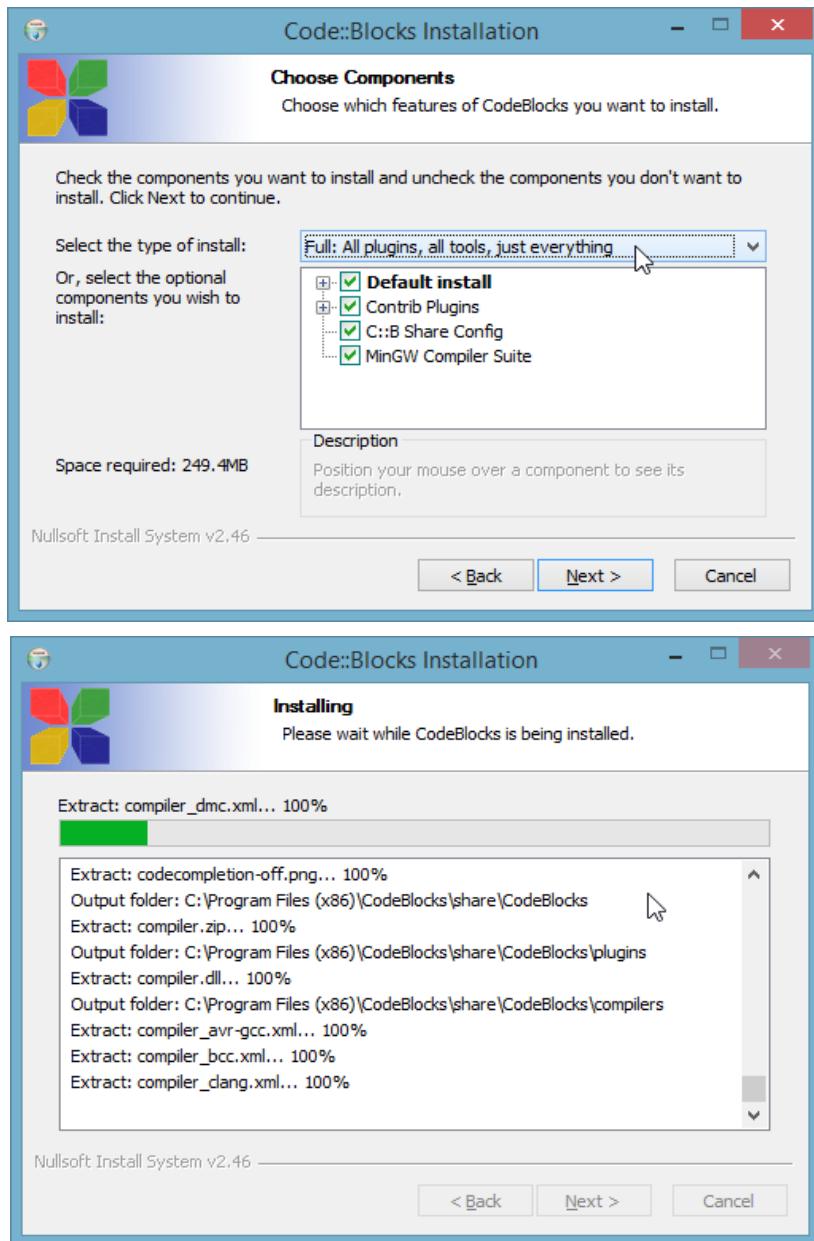
Windows 2000 / XP / Vista / 7:

File	Date	Download from
codeblocks-13.12-setup.exe	27 Dec 2013	BerliOS or Sourceforge.net
codeblocks-13.12mingw-setup.exe	27 Dec 2013	BerliOS or Sourceforge.net
codeblocks-13.12mingw-setup-TDM-GCC-481.exe	27 Dec 2013	BerliOS or Sourceforge.net

NOTE: The codeblocks-13.12mingw-setup.exe file includes the GCC compiler and GDB debugger from TDM-GCC (version 4.7.1, 32 bit). The codeblocks-13.12mingw-setup-TDM-GCC-481.exe file includes the TDM-GCC compiler, version 4.8.1, 32 bit. While v4.7.1 is rock-solid (we use it to compile C:B), v4.8.1 is provided for convenience, there are some known bugs with this version related to the compilation of Code::Blocks itself.

IF UNSURE, USE "codeblocks-13.12mingw-setup.exe"

اضغط على زر تحميل ملفات التشغيل لظهور لك صفحة تحتوي 3 خيارات للتحميل، قم بتنزيل الخيار الذي يحمل اسم codeblocks-13.12mingw-setup والذي يعني أن برنامج CodeBlocks مضافق إليه جميع ملفات المترجم gcc-compiler (قد يختلف الرقم 12-13 الموجود في اسم البرنامج إذا تم إصدار نسخة جديدة).
بعد الانتهاء من التحميل ابدأ بتنصيب الملف وتأكد من وضع علامة "صح" على جميع خيارات التنصيب الخاصة بالبرنامج (هذه الخيارات تعني تنصيب جميع ملحقات برنامج CodeBlocks كما في الصور التالية):



تنصيب برنامج WinAVR

بعد الانتهاء من تنصيب برنامج CodeBlocks سنقوم بتنزيل الـ WinAVR Toolchain الذي تحتوي على المترجم مفتوح المصدر AVR-GCC بالإضافة إلى جميع المكتبات البرمجية + برنامج avrdude

<http://sourceforge.net/projects/winavr/files>

اضغط على رابط التحميل الموجود أعلى الصفحة كما في الصورة التالية:



SOURCEFORGE

[Browse](#)
[Enterprise](#)
[Blog](#)
[Help](#)
Job

SOLUTION CENTERS
Go Parallel
Resources
Newsletters

Home / Browse / Development / Build Tools / WinAVR / Files

WinAVR

Brought to you by: [arcanum](#), [joerg_wunsch](#), [sprintersb](#)

[Summary](#) | [Files](#) | [Reviews](#) | [Support](#) | [Wiki](#) | [News](#) | [Code](#) | [Mailing Lists](#)

Looking for the latest version? [Download WinAVR-20100110-install.exe \(28.8 MB\)](#)

Home

/WinAVR/20100110/WinAVR-20100110-install.exe: released

Name	Modified	Size	Downloads / Week
WinAVR	2010-01-20		3,590
Release Candidate	2009-03-07		4

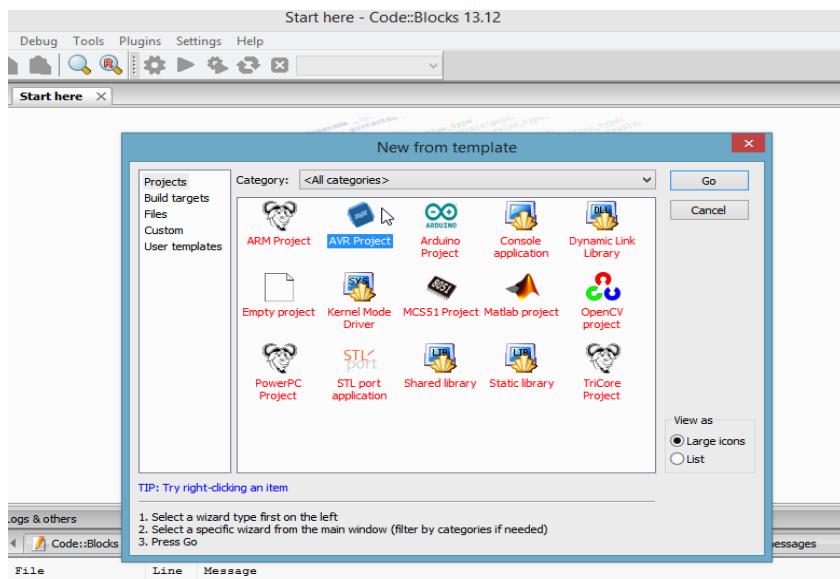
Totals: 2 Items

بعد الانتهاء من التحميل قم بتنصيب البرنامج كما في الخطوات السابقة. والآن سنقوم بتنزيل برنامج الواجهة الرسومية AVRDUDESS (إذا لم تكن قمت بتحميلها مسبقاً) تذكر أن برنامج `avrduude` لا يعمل إلا من خلال سطر الأوامر فقط لذا سنحتاج [AVRDUDESS](#) .

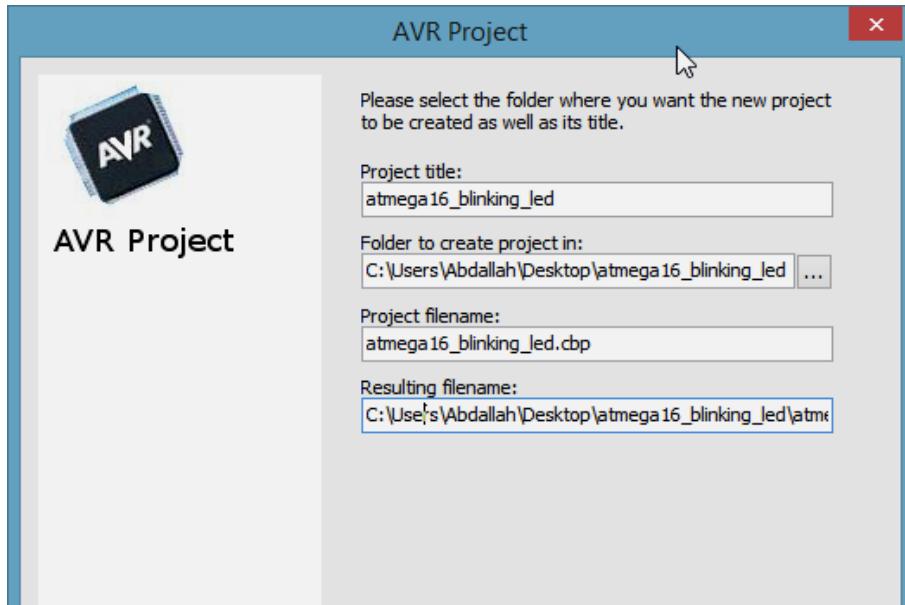
<http://blog.zakkemble.co.uk/avrdudess-a-gui-for-avrdude/>

ملاحظة: توفر العديد من البرامج الأخرى التي تعمل كواجهة رسومية غير AVRDUDESS مثل: `avrdude-gui` أو `BitBurner` أو `AVR8 Burn-O-Mat` أيضاً هناك العديد من البرامج الأخرى غير `avrduude` متوفرة لنظام ويندوز لكنني فضلت `avrdude` لأنه يأتي مع `avrtoolchain` ويعمل على جميع أنظمة التشغيل

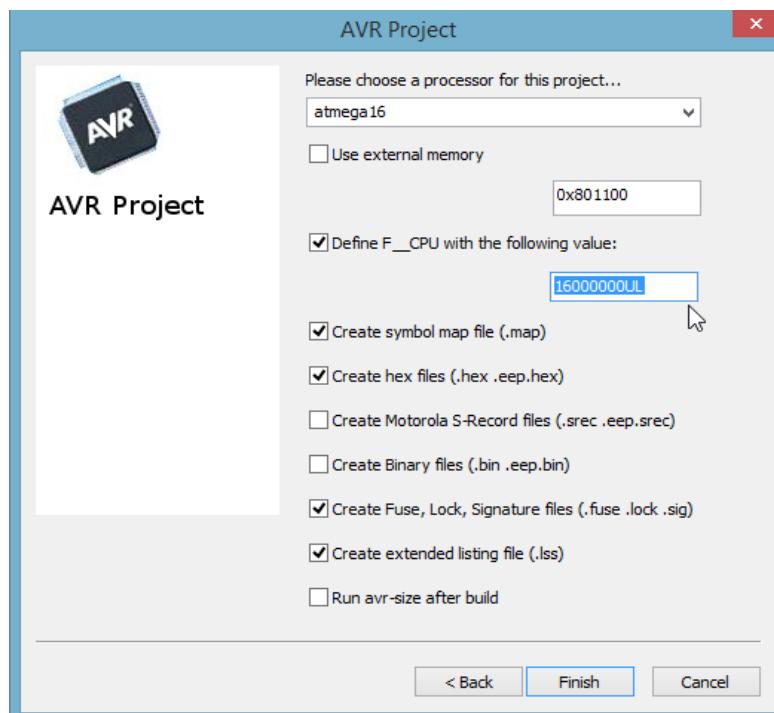
الآن يمكنك البدء في كتابة البرامج لمحكمات AVR ، لتأخذ الدا Blinking Led كمثال قم بتشغيل برنامج `CodeBlocks` واختر `New Project` ومن الصفحة التي ستظهر اختر AVR project كما في الصور التالية:



قم بكتابة اسم المشروع `ATmega16_blinking_led` واختر المكان الذي تريده أن تحفظ به ملفات المشروع كما في الصورة التالية.

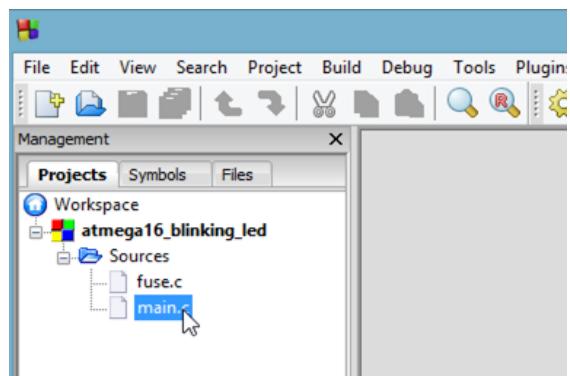


في الصفحة التالية قم بالتأشير على خيار `Create Release` فقط وألغ التأشير على `Create Debug` configuration. ستظهر صفحة جديدة خاصة بإعدادات المُتحكِّم الدقيق، من القائمة العلوية اختر نوع المُتحكِّم `ATmega16` ثم قم بتغيير سرعة تشغيل المعالج إلى التردد المطلوب مثل 8 ميجا هرتز بدل من 16 ميجا وذلك عبر كتابة الرقم `8000000`



الآن أصبح كل شيء جاهز لنكتب أول برنامج، حيث سنجد أن برنامج CodeBlocks قام بعمل ملفين الأول يسمى main.c و الثاني هو fuses.c – سنتعامل مع الملف الأول فقط وهو الملف الذي سنكتب فيه البرامج الخاصة بالتحكم، أما الفيوزات فيمكنك أن تقوم بإعدادها مباشرة كما هو مذكور في فصل الفيوزات.

للبدء قم بالضغط مررتين على ملف main.c من القائمة الجانبية للبرنامج لظهور الصفحة الخاصة بكتابة الكود البرمجي وبعد الانتهاء من كتابة الكود اضغط على زر من الشريط العلوي للبرنامج (الزر الذي يمتلك رمز الترس) كما في الصورة التالية:





main.c [atmega16_blinking_led] - Code::Blocks 13.12

File Debug Tools Plugins Settings Help

main.c X Build

```

1  /*
2   Hello AVR World !
3   Make a Blinking led with atmega16 and led on PORTC(pin PC0)
4   */
5
6   #include <avr/io.h>
7   #include <avr/delay.h>
8
9   int main(void)
10 {
11     DDRC = 0b11111111;
12
13     while(1)
14     {
15       PORTC = 0b00000000;
16       _delay_ms(500);
17
18       PORTC = 0b00000001;
19       _delay_ms(500);
20
21     }
22
23     return 0;
24
25 }
```

ستظهر في القائمة السفلية للبرنامج عبارة تدل على نجاح عملية التجميع Compilation errors بدون أخطاء . الآن يصبح لدينا ملف الـ hex الذي يمكننا استخدامه إما لبرمجة المُتحكِّم أو يمكننا استخدامه مع برنامج المحاكاة بروتس Protues لعمل محاكاة للمُتحكِّم ATmega16

Code::Blocks X Search results X Build log X Build messages X Debugger X

File Line Message

```

/usr/lib/avr/include... 95 == Build: Debug in atmega16 (compiler: GNU GCC Compiler for AVR) ===
main.c      warning: #warning "Compiler optimizations disabled; functions from <util/delay.h> won't work as designed" [-Wcpp]
main.c      9  warning: unused variable 'sweeper' [-Wunused-variable]
== Build finished: 0 error(s), 2 warning(s) (0 minute(s), 1 second(s)) ===
```



ملحق: ترجمة الملفات باستخدام makefile

تُعد هذه الطريقة هي أسرع اسلوب لترجمة الملفات المكتوبة بلغة السي وتحويلها إلى الصيغة التقنية مثل ملفات hex (أو حتى ترجمة البرامج التقليدية الخاصة بالحاسوب الآلي).

تعتمد هذه الطريقة على استخدام المترجم avr-gcc مباشرة دون الحاجة لوجود أي بيئة تطوير مثل Atmel studio أو Codeblocks. كما تمتاز بأنها معيارية وصالحة للعمل على جميع نظم التشغيل windows, linux, mac وبسبب عدم الحاجة لوجود IDE نجد أن هذه الطريقة هي الأخف على الإطلاق لذا لا تستعجب إذا وجدت أن معظم المشاريع الموجودة على الإنترنت تستخدم هذه الطريقة في الترجمة.

شخصياً أستخدم هذا الأسلوب في جميع المشاريع التي أعمل عليها حيث استخدم محرر النصوص NotePad ++ أو sublime لكتابة الملفات بلغة السي ثم أحولها إلى ملفات hex باستخدام الـ makefile (سواء على ويندوز أو لينكس).

التجربة الأولى

في البداية تأكد من وجود الـ toolchain على جهازك (على نظام ويندوز تسمى WinAVR toolchain) ويمكن تحميلها من الرابط التالي

<http://sourceforge.net/projects/winavr/files>

- لاختبار تنصيب الـ toolchain بصورة صحيحة، قم بفتح سطر الأوامر واتبع الأمر
- سطر الأوامر على نظام ويندوز يسمى command prompt ويتم تشغيله بكتابة بالأمر cmd من قائمة start
 - سطر الأوامر على نظام لينكس يسمى terminal (الطرفية)

إذا كانت الـ toolchain منصبة بصورة صحيحة يفترض أن تظهر الرسالة التالية

make: *** No targets specified and no makefile found. Stop.

```
c:\>make
make: *** No targets specified and no makefile found. Stop.
c:\>
```

والآن لنقم بتجربة الملف الأول وهو المثال led blinking compile with make file، قم بفتح المجلد المسمى compile with make file لتجد بداخله ملفين وهما

main.c
makefile



الملف main.c يحتوي على المثال الأول في الكتاب blinking led أما الملف `makefile` فيحتوي على جميع الإعدادات الخاصة بتحويل الملف إلى ملف الهيكس. قم بفتح الملف باستخدام أي محرر نصوص مثل `notepad++`

```

1 ##### Target Specific Details #####
2 ##### Customize these for your project #####
3
4 # Name of target controller
5 # (e.g. 'at90s8815', see the available AVR-GCC MCH
6 # options for possible values)
7 MCU=atmega16
8
9 # id to use with programmer
10 # default: PROGRAMMER MCU=$MCU
11 # In case the PROGRAMMER used, e.g. avrdude, doesn't
12 # accept the same MCU name as AVR-GCC (for example
13 # for ATmega8s, AVR-GCC expects 'atmega16' and
14 # avrdude requires 'm16')
15
16 PROGRAMMER MCU=m16
17
18 # Name of our project
19 # (use a single word, e.g. 'myproject')
20 PROJECTNAME=blinkingled
21
22 # Source files
23 # List C/C++/Assembly source files:
24 # (list all files to compile, e.g. 'a.c b.cpp as.S'):
25 # Use .cc, .cpp or .C suffix for C++ files, use .S
26 # (NOT .s !!!) for assembly source code files.
27 PRJSRC=main.c

```

يمتلك هذا الملف الكثير من الإعدادات وأهمها هي المجموعة التالية:

MCU=ATmega16

هنا يتم وضع اسم المُتحكم الدقيق المستخدم والذي سيتم توليد ملف الهيكس خصيصاً له (مع ملاحظة أن ملفات الهيكس تختلف من مُتحكم لأخر).

PROGRAMMER MCU=m16

هذا الخيار يحدد اسم المُتحكم الذي سيرفع عليه ملف الهيكس باستخدام برنامج avrdude ويجب تسميته بالحرف m ثم رقم المُتحكم فمثلاً

- ATmega16 = m16
- ATmega32 = m32
- atmega328p = m328p

PROJECTNAME=blinkingled

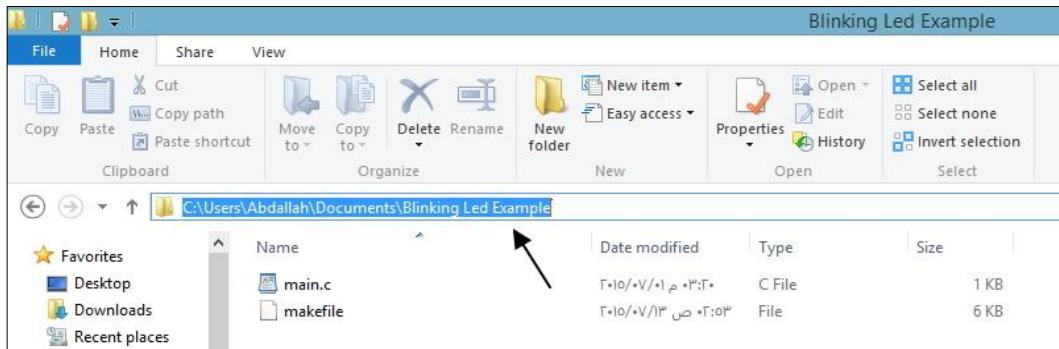
هذا هو اسم المشروع وسيكون اسم ملف الهيكس الذي سيتم توليد

PRJSRC=main.c

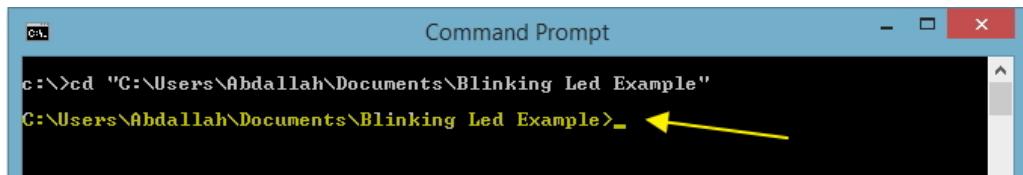
اكتب هنا اسم الملف الذي يحتوي على الكود البرمجي المراد تحويله إلى ملف هيكس.

احفظ الملف ثم توجه إلى سطر الأوامر وحول مسار المجلد الحالي إلى نفس مكان الملفين main.c و makefile فمثلاً- المسار الذي استخدمه هو

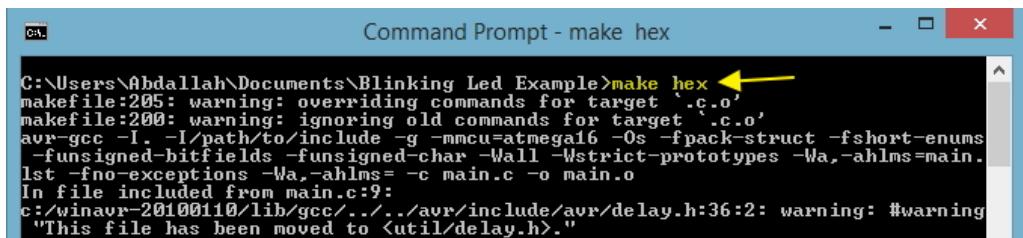
C:\Users\Abdallah\Documents\Blinking Led Example



قم بكتابة الأمر `cd "path"` واستبدل كلمة `path` بالمسار ثم اضغط `Enter` لتجد أن سطر الأوامر انتقل إلى المجلد المطلوب كما في الصورة التالية



والآن اكتب الأمر `make hex` لتجد أن المترجم بدأ عملية تحويل الملف `main.c` إلى الملف `blinkingled.hex` كما في الصورة التالية



شكل المجلد بعد أن تم توليد الملف `blinkingled.hex`



C > Documents > Blinking Led Example			
Name	Date modified	Type	Size
blinkingled.ee.hex	٢٠١٥/٠٧/١٣ ٠٣:١١ ص	HEX File	1 KB
blinkingled.hex	٢٠١٥/٠٧/١٣ ٠٣:١١ ص	HEX File	1 KB
blinkingled.out	٢٠١٥/٠٧/١٣ ٠٣:١١ ص	OUT File	4 KB
blinkingled.out.map	٢٠١٥/٠٧/١٣ ٠٣:١١ ص	Linker Address Map	10 KB
main.c	٢٠١٥/٠٧/١٣ ٠٣:١٠ م	C File	1 KB
main.o	٢٠١٥/٠٧/١٣ ٠٣:١١ ص	O File	4 KB
makefile	٢٠١٥/٠٧/١٣ ٠٣:٥٣ ص	File	6 KB

مسح الملفات من خلال make

بافتراض أنك تريدين مسح جميع الملفات التي نتجت من عملية الترجمة فيمكنك ذلك بسهولة باستخدام الأمر `make clean`

ملاحظة: ملف `makefile` المرفق مع الكتاب هو نسخة مبسطة من هذا النوع من الملفات وقد تجد في مشاريع أخرى نفس

الملف ولكن بإعدادات أكثر كما سنرى في الفصل الخاص بـ Real Time Operating system

يستحسن أن تعتاد وتنتقن ترجمة الملفات بهذه الطريقة لأنها تعتبر الطريقة المعتمدة في مجتمعات المطوريين وستجد الكثير من المشاريع على الإنترنت تستخدم هذه الطريقة.

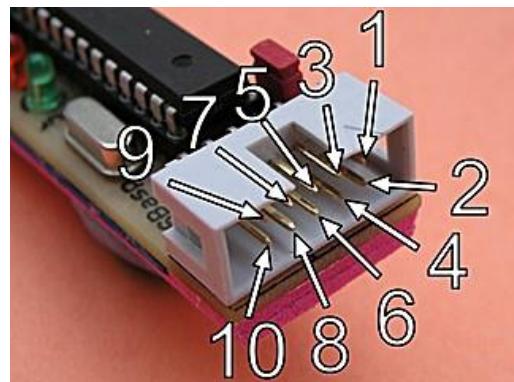
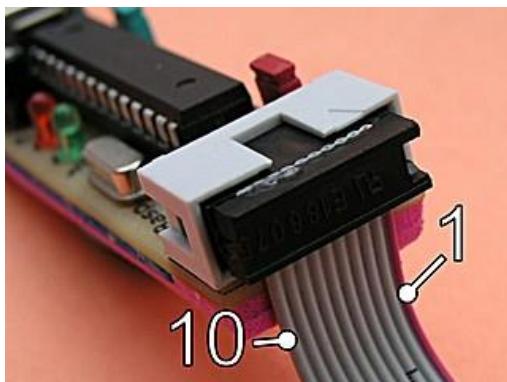
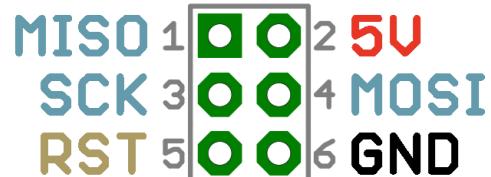
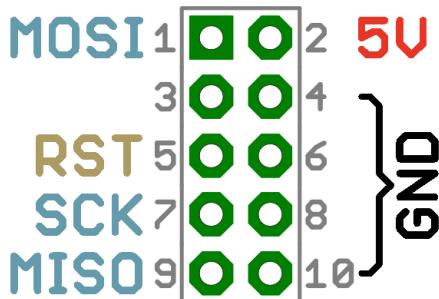


مُلْحِق: رفع ملف الـ Hex على المُتَحَكِّم الدَّقِيق

كيف توصل المبرمجة بالمتتحكم

تمتلك أغلب مبرمجات AVR إما 6 أو 10 أطراف (تُسمى هذه الأطراف ISP connector) وتكون مرتبة كما هو موضح بالصورة التالية

مثال على ذلك المبرمجة USBasp



لبرمجة المُتَحَكِّم الدَّقِيق كل ما عليك فعله هو توصيل كل طرف في المبرمجة بما يوازيه في المُتَحَكِّم الدَّقِيق وهذا يشمل الأطراف

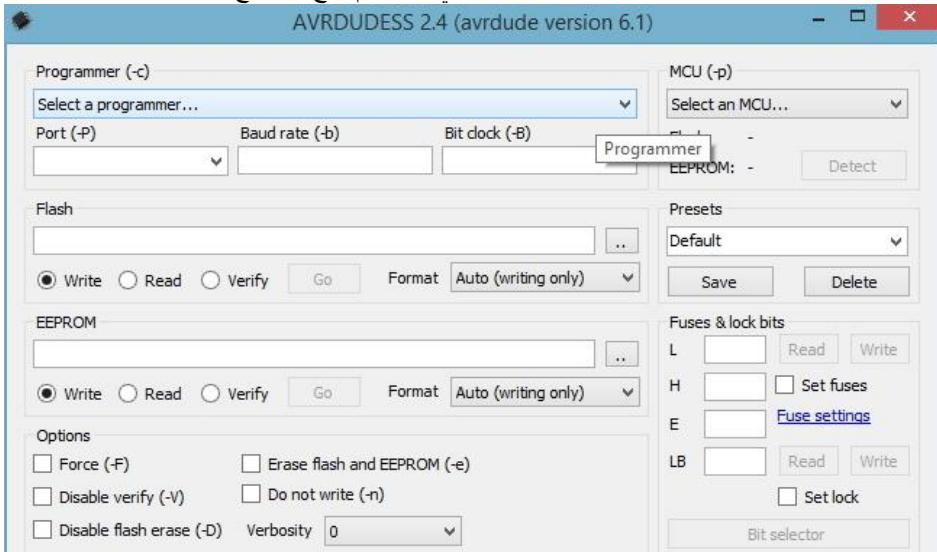
MOSI - MISO - SCK - RST(RESET)

والطرف 5V في المبرمجة يتصل بالـ VCC مع AVCC وكذلك يتم توصيل كل اطراف الـ GND ببعضها (المُتَحَكِّم والمبرمجة).

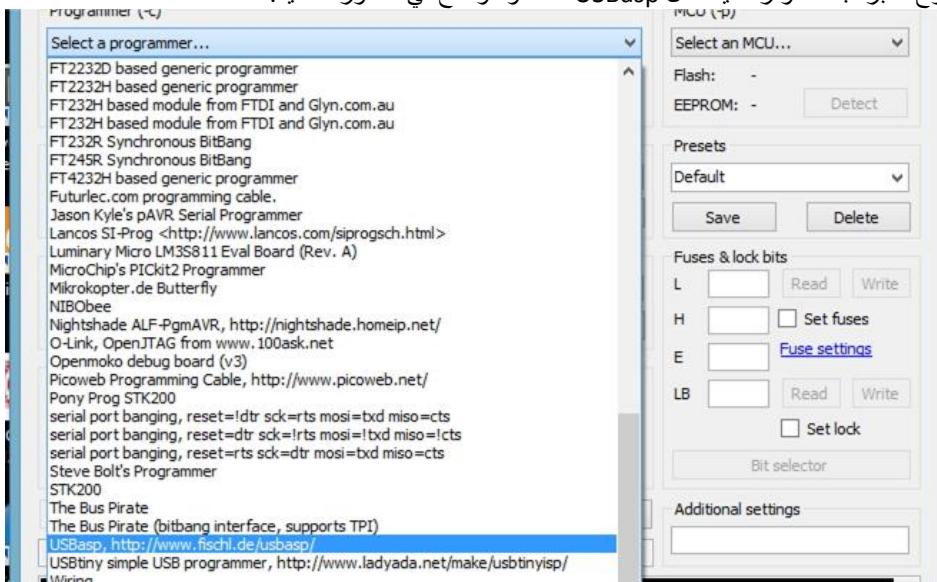
(XCK/T0) PB0	1	40	PA0 (ADC0)
(T1) PB1	2	39	PA1 (ADC1)
(INT2/AIN0) PB2	3	38	PA2 (ADC2)
(OC0/AIN1) PB3	4	37	PA3 (ADC3)
(SS) PB4	5	36	PA4 (ADC4)
(MOSI) PB5	6	35	PA5 (ADC5)
(MISO) PB6	7	34	PA6 (ADC6)
(SCK) PB7	8	33	PA7 (ADC7)
RESET	9	32	AREF
VCC	10	31	GND
GND	11	30	AVCC
XTAL2	12	29	PC7 (TOSC2)
XTAL1	13	28	PC6 (TOSC1)
(RXD) PD0	14	27	PC5 (TDI)



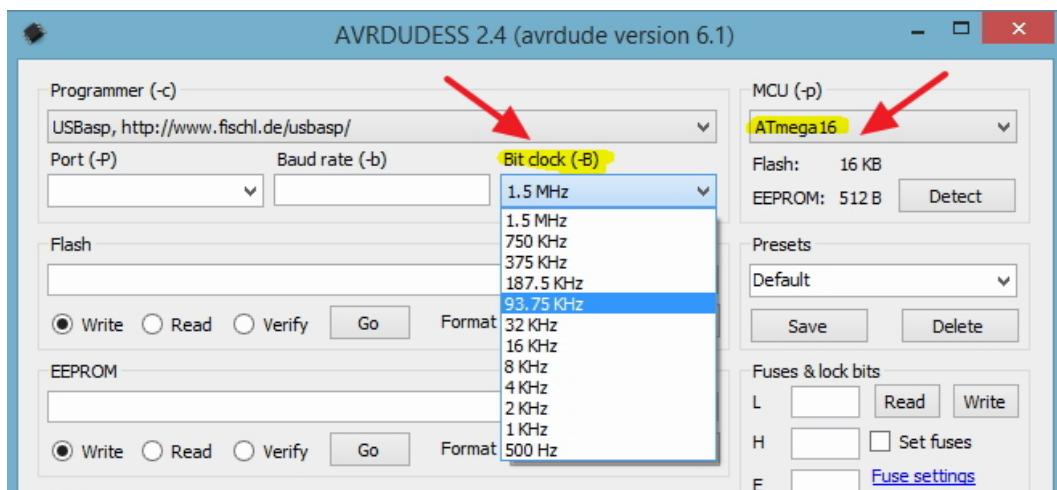
بعد الانتهاء من توصيل دائرة المتحكم على لوحة التجارب Breadboard سنقوم برفع ملف الهيكس باستخدام برنامج AVRdude وذلك من خلال الواجهة الرسمية AVRdude. في البداية قم بفتح البرنامج



اختر نوع المبرمجة المتوفرة لديك مثل USBasp كما هو موضح في الصورة التالية:

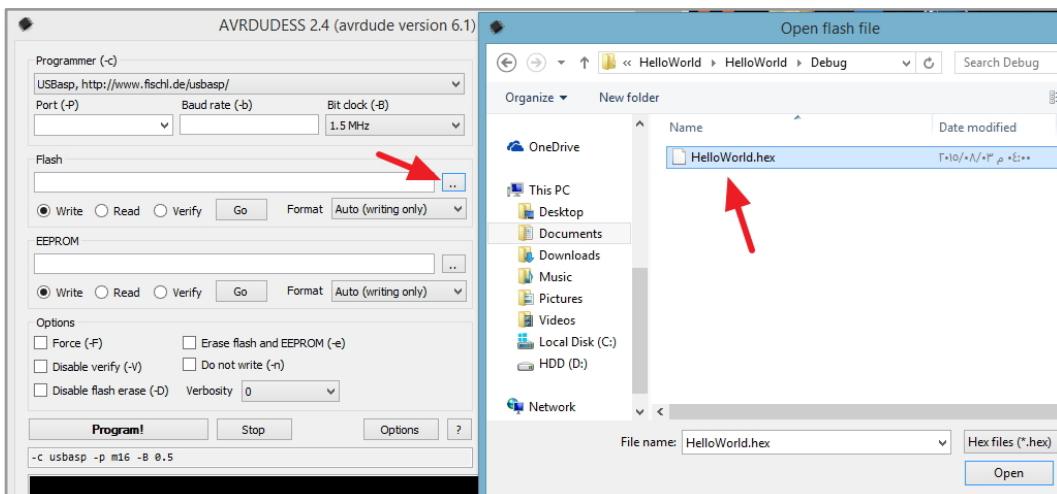


بعد اختيار المبرمجة قم باختيار نوع المتحكم الدقيق (من القائمة الموجودة على الجانب الأيمن من البرنامج) وفي حالة أن المتحكم يعمل بدائرة المذبذب الداخلي بسرعة **1 ميجا** يجب أن تغير سرعة رفع البرنامج لتصبح **93 كيلوبايت في الثانية** أما إذا كان المتحكم يعمل باستخدام دائرة مذبذب خارجي مثل الكريستال 16 ميجا في يمكنك أن تترك خيار سرعة الرفع يساوي الخامس ميجا كما هو موضح في الصورة التالية.



ملاحظة: يستطيع برنامج AVRdudess أن يتعرف على المتحكم بصورة تلقائية وذلك عبر الضغط على زر Detect الموجود على جانب الشاشة الأيسر.

والآن اختر ملف الهيكس الذي يحتوي على البرنامج المطلوب رفعه إلى المتحكم الدقيق



وأخيراً قم بالضغط على زر Program الموجود أسفل شاشة البرنامج

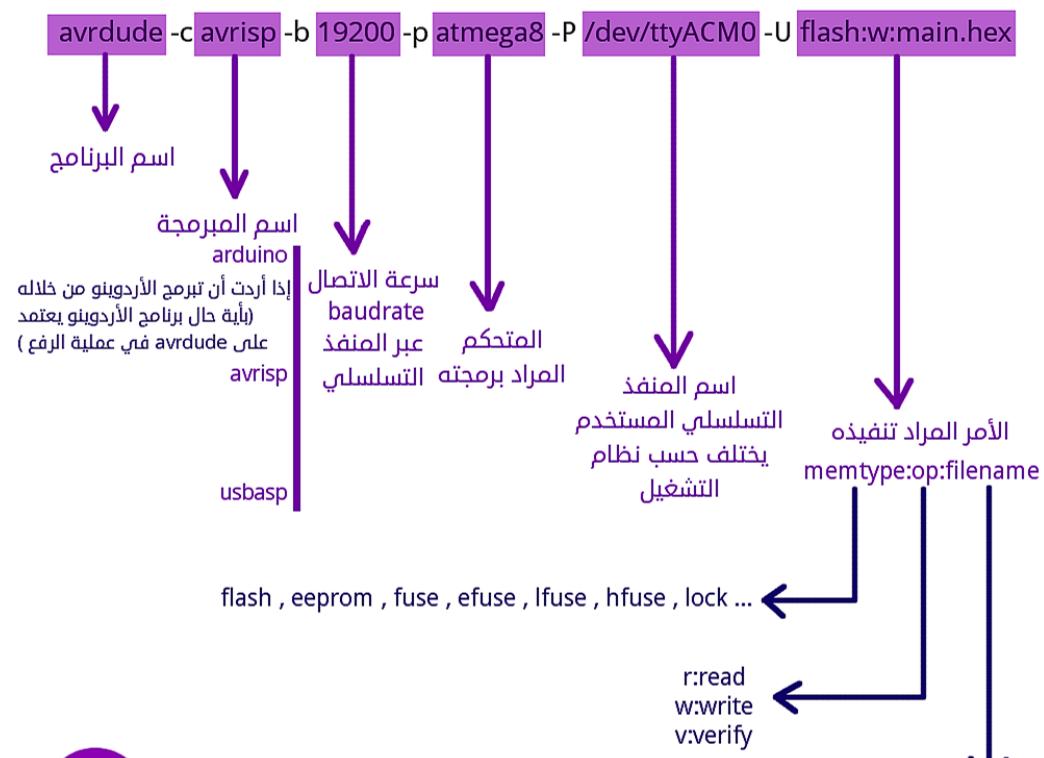


استخدام برنامج avrdude بدون واجهة رسومية

بالرغم من وجود العديد من الواجهات الرسومية لبرنامج avrdude إلا أنه بالأساس يعمل من خلال سطر الأوامر (سواء على ويندوز أو على shell على نظام لينكس).

قد يبدو استخدام avrdude مزعجاً أو مخيفاً لمن لا يستخدم نظام لينكس وليس على احتكاك بسطر الأوامر إلا أنه سنشكّل سوية مدى سهولة استخدامه.

الشكل العام لتعليمات avrdude



أهم الخيارات المتاحة

اسم المبرمجة : بكتابه c- ك الخيار في سطر الأمر avrdude لتحديد نوع المبرمجة مثلاً : arduino – avrISP – usbasp – .. الخ.



المُتحكم العراد برمجته : بكتابه p- ك الخيار في سطر الأمر avrdude لتحديد اسم المُتحكم.

سرعة الاتصال : بكتابه b- ك الخيار في سطر الأمر ، و هذا الخيار مهم لتأكيد كون سرعة الاستقبال في الكود المنفذ على مُتحكم "المُبرمج" يساوي سرعة avrdude لأنَّه اختلاف السرعتين سيؤدي إلى أخطاء في التزامن.

المنفذ التسلسلي : بكتابه P- ك الخيار في سطر الأمر . اسم المنفذ يتختلف من نظام تشغيل لآخر فغالباً ما يكون من نمط ACM بالنسبة للينكس و COM بالنسبة للويندوز .

الأمر التنفيذي : بكتابه l- ك الخيار في سطر الأمر ، و من ثم تحديد الأمر بالصياغة التالية : format memtype:op:filename:format و يقصد بـ format نوع الملف المستخدم حيث يمكن استخدام الملفات من نوع hex أو bin. أما filename فيتم استبدالها بأسم الملف، وكلمة op تعني operation وهي العملية المطلوب تنفيذها مثل read (اقرأ الذاكرة) أو write (أكتب في الذاكرة) أو verify والتي تعني التأكد من أن محتوى الذاكرة يطابق ملف الميكس الذي يتم تحديد أسمه في نفس الأمر.

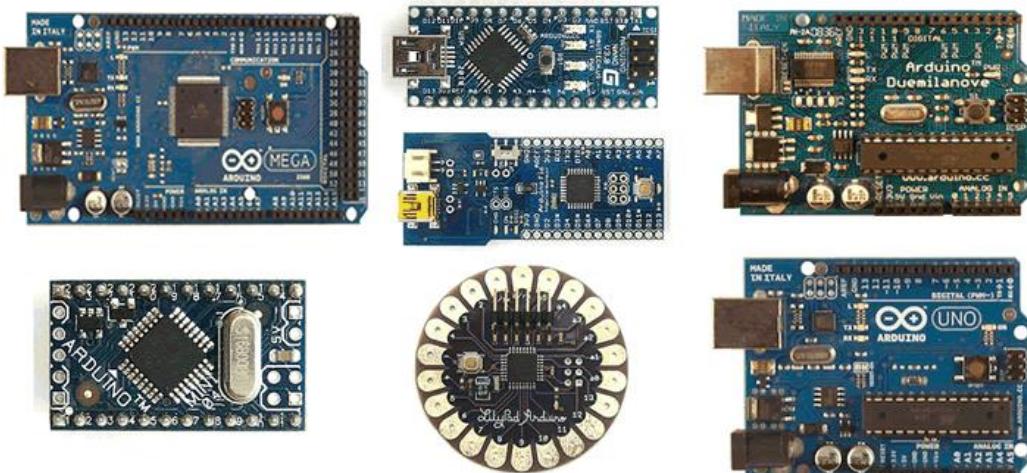
خيارات مختلفة : من الظلم إختصار خيارات avrdude بهذه الأسطر القليلة و لكل خيار حالة استخدام خاصة و هي مشرورة بوضوح في كتيب البرنامج و يمكن الاطلاع عليها من خلال كتابة man avrdude على سطر الأوامر في لينكس .

المقال السابق عن استخدام برنامج avrdude من سطر الأوامر منقول من موقع عنديات تحت رخصة المشاع الإبداعي
CC-BY-SA-NC



مُلْحِق: كِيف تُسْتَخِدُ لُوْحَاتَ آرْدُوِينُو لِتَعْلِمُ بِرْمَجَةَ AVR

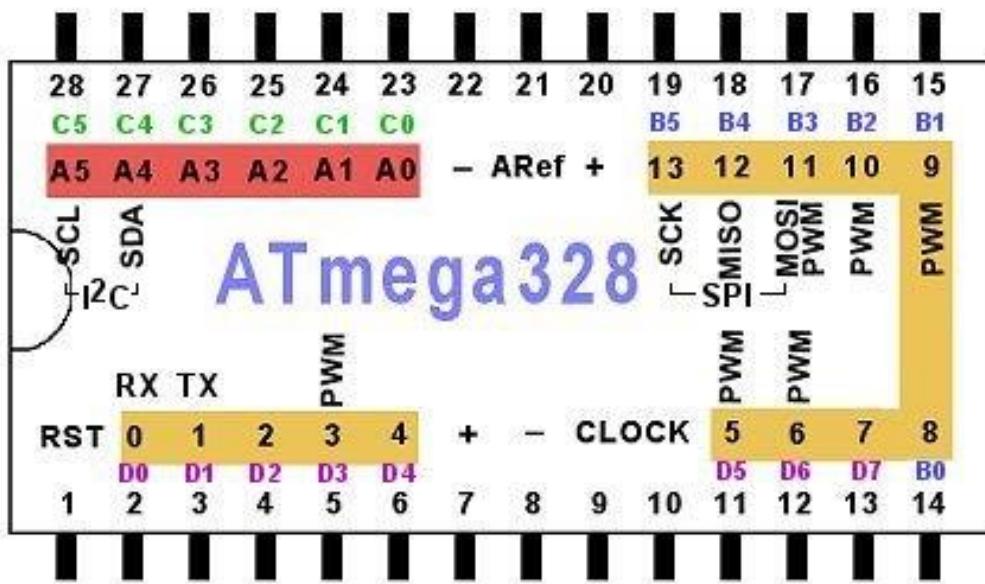
تُعد لُوْحَاتَ آرْدُوِينُو Arduino من أرْخُصِ اللُّوْحَاتِ النَّطَوِيِّيَّةِ فِي الْعَالَمِ حِيثُ تَبْدَأُ أَسْعَارُهَا مِنْ 5 دُولَارٍ فَحَسْبٍ مُّثُلَ لُوْحَةَ Arduino nano. وَمَا يَمْيِزُ هَذِهِ اللُّوْحَاتِ أَنَّهَا تَعْمَلُ بِمُتَحَكِّمَاتِ AVR وَبِالْتَّحْدِيدِ عَائِلَةَ Atmega (هُنَّاكَ لُوْحَاتُ آرْدُوِينُو تَعْمَلُ بِعَائِلَةَ ATTiny أَيْضًا).



الْحَقِيقَةُ أَنَّ بِرْنَامِجَ Arduino IDE مَاهُو إِلَّا الْمُتَرَجِّمُ الشَّهِيرُ AVR-GCC الَّذِي نَسْتَخْدِمُهُ فِي هَذَا الْكِتَابَ+الْوَاجِهَةِ الرَّسُومِيَّةِ الْخَاصَّةِ بِبِرْنَامِجِ Processing وَمَضَافٍ إِلَيْهِ الْعَدِيدُ مِنَ الْمَكْتَبَاتِ الْبَرْمَجِيَّةِ مِنْ مَشْرُوْعِ Wiring + بَعْضِ التَّعَدِيلَاتِ الْبَسيِطَةِ عَلَى بِرْنَامِجِ avrdude. وَهَذَا يَجْعَلُ بِرْنَامِجَ آرْدُوِينُو مُتَوَافِقًا تَامًا مَعَ الْبِرَّامِجِ الْمَكْتُوبَةِ بِلُغَةِ C-ANSI.

هَذَا أَمْرٌ وَاحِدٌ يَجْبُ الْإِنْتِبَاهُ لَهُ عِنْدَ التَّعَالِمِ مَعَ لُوْحَاتَ آرْدُوِينُو وَهُوَ تَرْقِيمُ الْأَطْرَافِ، حِيثُ نَجِدُ أَنَّ مُصْمِمِي لُوْحَاتَ آرْدُوِينُو لَدِيهِمْ أَسْلُوبٌ مُخْتَلِفٌ لِتَرْقِيمِ أَطْرَافِ مُتَحَكِّمَاتِ Atmega المُوْجَدَةِ عَلَى اللُّوْحَاتِ وَلَا يَتَمَّ اسْتِخْدَامُ أَسْمَاءِ الْبُورَتَاتِ مُثُلَ 0,1,2,3 port A, port B وَإِنَّمَا يَتَمَّ اسْتِخْدَامُ تَرْقِيمٍ بِسَيِطٍ مُثُلَ 0,1,2,3

عَلَى أَيِّ حَالٍ هَذَا الْأَمْرُ لَا يَمْثُلُ مُشَكَّلَةً فَكُلُّ مَا عَلَيْكَ مَعْرِفَتُهُ هُوَ أَسْمَاءُ الْأَطْرَافِ عِنْدَ بِرْمَجَتِهَا. لَنَأْخُذُ لُوْحةَ آرْدُوِينُو Uno كَمَثَلٍ (بِاعْتِبَارِهَا أَشْهَرُ لُوْحَاتَ آرْدُوِينُو). الصُّورَةُ التَّالِيَّةُ تَمْثُلُ تَرْقِيمَ أَطْرَافِ الْمُتَحَكِّمِ ATmega328 بِكُلِّ مِنَ الْأَسْلُوبِ الْأَصْلِيِّ (مُثُلُ مَا هُوَ مَذَكُورُ فِي دَلِيلِ الْبَيَانَاتِ + تَرْقِيمِ آرْدُوِينُو).



Digital Input/Output

Analog / Digital

كما نرى في الصورة السابقة نجد أن الترقيم المكتوب في المربعات البرتقالية والحراء هو ترقيم آردوينو بينما الترقيم المكتوب بالحروف الزرقاء هو الترقيم الأصلي للأطراف ويعبر عن اسم البورت مثل D0 تعني 0 port D pin 0.

والآن لنقم بكتابه برنامج Blinking Led على طريقة الـ C – ANSI

ملاحظة: تحتوي معظم لوحات آردوينو على دايدون ضوئي متصل بالطرف رقم 13 وهو في الحقيقة الطرف PB5 (البورت B – الطرف الخامس).

قم بفتح برنامج IDE Arduino وأكتب برنامج الـ Blinking led كما هو موضح في المثال الأول في الفصل الثالث من الكتاب. مع تغيير بسيطة وهو تعريف سرعة المعالج برقم 16 ميجا

```
#define F_CPU 16000000UL
```

بعد الانتهاء من كتاب البرنامج قم بالضغط على زر Upload لتجد أن برنامج Arduino قام بترجمة الملف وتحويله إلى Hex file كما هو موضح بالصورة التالية



sketch_aug23a | Arduino 1.6.5

File Edit Sketch Tools Help

sketch_aug23a

```

1 #define F_CPU 16000000UL
2 #include <avr/io.h>
3 #include <avr/delay.h>
4
5 int main(void)
6 {
7
8     DDRB |= (1<<PB5);
9
10    while(1)
11    {
12        PORTB |= (1<<PB5);
13        _delay_ms(500);
14
15        PORTB &= ~(1<<PB5);
16        _delay_ms(500);
17    }
18    return 0;
19 }
20

```

Done compiling.

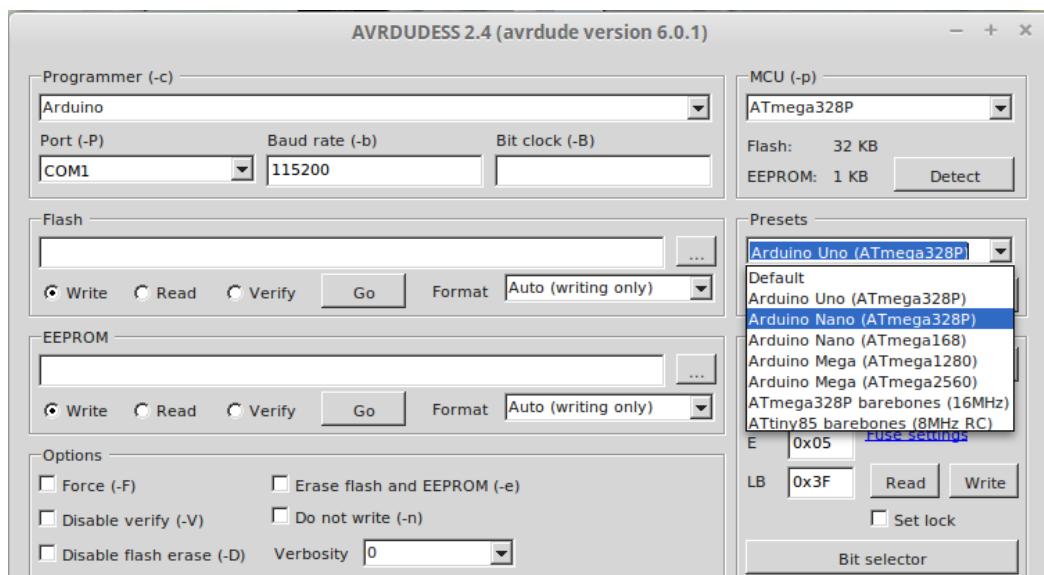
Sketch uses 176 bytes (0%) of program storage space. Maximum is 30,720 bytes.
 Global variables use 0 bytes (0%) of dynamic memory, leaving 2,048 bytes for local variables. Maximum is 2,048 bytes.

إذا لم تكن تفضل استخدام برنامج آردوينو في كتابة الكود فيمكنك أن تستخدم برنامج Atmel Studio مع إضافة الدعم المباشر للوحات آردوينو بسهولة من خلال إضافة visualmicro والتي تجعل برنامج Atmel studio يتعامل مع لوحات آردوينو مباشرة

<http://www.visualmicro.com/page/Arduino-for-Atmel-Studio.aspx>



أيضاً يمكنك استخدام ملفات `makefile` في ترجمة الأكواد البرمجية وتحويلها إلى ملف هيكس (راجع ملحق شرح `ARVdudess`) ثم استخدام برنامج `ARVdudess` لرفعها على لوحات آردوينو (`makefile`)



أيضاً بامكانك استخدام لوحات آردوينو كمبرمجة AVR لاي متحكم Programmer وذلك عبر برنامج AVRdudess

ملاحظة: بامكانك أيضاً كتابة برامج بلغة الأسمبلي داخل برنامج Arduino IDE و التجربة التالية تشرح برنامج Blinking Led بلغة الأسمبلي

<https://ucexperiment.wordpress.com/2013/05/31/arduino-blink-using-gcc-inline-assembly>



قائمة المراجع

مراجع تعليمية عربية

دورة الإلكترونيات العملية د. وليد عيسى (أساسيات الإلكترونيات من الصفر)

<https://www.youtube.com/playlist?list=PLww54WQ2wa5rOJ7FcXxi-CMNgmpybv7ei>

دورة المهندس وليد بليد في شرح برمجة متحكمات AVR باستخدام لغة Bascom

<https://www.youtube.com/playlist?list=PLww54WQ2wa5qWSTU7MYqjN0jHNaWuoXUK>

قناة شركة ENG Unity وتحتوي على العديد من الدورات العربية عن النظم المدمجة ويتضمن ذلك AVR و برنامج Digital Circuits والتصميم الرقمي Altium

<https://www.youtube.com/user/ENGUnity/playlists>

قناة عربية تحتوي على دورات فيديو مبسطة عن النظم المدمجة وتنضم

C for Embedded System
Micrcontroller Architectre

<https://www.youtube.com/channel/UCbZ7PLd5LAnje1hpyoiRW0A/playlists>

دورة تعلم برمجة AVR باستخدام برنامج CodeVision

<http://www.qariya.info/vb/showthread.php?t=81782>

موقع "عثاديات" يحتوي على مجموعة من المقالات المبسطة

<http://www.atadiat.com/%D8%A7%D9%84%D9%82%D8%B3%D9%85-%D8%A7%D9%84%D8%AA%D8%B9%D9%84%D9%8A%D9%85%D9%8A/>



مراجع تعليمية إنجليزية

مراجع AVR العملاق (يعد من أفضل المراجع في العالم للنظم المدمجة)

AVR Microcontroller and Embedded Systems: Using Assembly and C (Pearson Custom Electronics Technology) - Muhammad Ali Mazidi

<http://www.amazon.com/AVR-Microcontroller-Embedded-Systems-Electronics/dp/0138003319>

من أفضل المراجع التي تعلمك "كيفية تحويل متطلبات العميل إلى أفضل تصميم برنامج على النظم المدمجة"
An Embedded Software Primer

<http://www.amazon.com/Embedded-Software-Primer-David-Simon/dp/020161569X>

دورة **NewBieHack** لتعلم AVR في 55 درس مفصل، وتعتبر من أفضل الدورات المبسطة والممتعة
<https://www.youtube.com/playlist?list=PLE72E4CFE73BD1DE1>

FreeRTOS المرجع الرسمي لنظام
Using the FreeRTOS Real Time Kernel - Standard Edition

<http://www.amazon.com/Using-FreeRTOS-Real-Time-Kernel/dp/1446169146/>